

00862.023110.



PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:)	
	:	Examiner: NOT YET ASSIGNED
MAKOTO MIHARA)	
	:	Group Art Unit: NOT YET ASSIGNED
Application No.: 10/600,843)	
	:	
Filed: June 23, 2003)	
	:	
For: METHOD, PROGRAM, AND)	
STORAGE MEDIUM FOR	:	
ACQUIRING LOGS)	September 24, 2003

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

SUBMISSION OF PRIORITY DOCUMENTS

Sir:

In support of Applicant's claim for priority under 35 U.S.C. § 119, enclosed are certified copies of the following foreign application:

2002-191128, filed June 28, 2002; and

2002-191129, filed June 28, 2002.

Applicant's undersigned attorney may be reached in our New York office by telephone at (212) 218-2100. All correspondence should continue to be directed to our address given below.

Respectfully submitted,

A handwritten signature in black ink, appearing to read "Frank J. Scinto", is written over a horizontal line.

Attorney for Applicant

Registration No. 42,476

FITZPATRICK, CELLA, HARPER & SCINTO
30 Rockefeller Plaza
New York, New York 10112-3801
Facsimile: (212) 218-2200

NY_MAIN 377077v1

CEMO311045
Appln. NO. 10/600,843
EAU: NYA

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日
Date of Application: 2 0 0 2 年 6 月 2 8 日

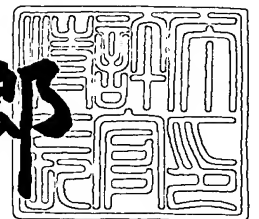
出 願 番 号
Application Number: 特 願 2 0 0 2 - 1 9 1 1 2 8
[ST. 10/C]: [J P 2 0 0 2 - 1 9 1 1 2 8]

出 願 人
Applicant(s): キヤノン株式会社

2 0 0 3 年 7 月 1 0 日

特許庁長官
Commissioner,
Japan Patent Office

太田信一郎



出証番号 出証特 2 0 0 3 - 3 0 5 5 5 9 6

【書類名】 特許願

【整理番号】 4741003

【提出日】 平成14年 6月28日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 9/00

【発明の名称】 ログ取得方法およびプログラム、記憶媒体

【請求項の数】 14

【発明者】

 【住所又は居所】 東京都大田区下丸子 3 丁目 3 0 番 2 号 キヤノン株式会社
社内

 【氏名】 三原 誠

【特許出願人】

 【識別番号】 000001007

 【氏名又は名称】 キヤノン株式会社

【代理人】

 【識別番号】 100076428

 【弁理士】

 【氏名又は名称】 大塚 康德

 【電話番号】 03-5276-3241

【選任した代理人】

 【識別番号】 100112508

 【弁理士】

 【氏名又は名称】 高柳 司郎

 【電話番号】 03-5276-3241

【選任した代理人】

 【識別番号】 100115071

 【弁理士】

 【氏名又は名称】 大塚 康弘

 【電話番号】 03-5276-3241

【選任した代理人】**【識別番号】** 100116894**【弁理士】****【氏名又は名称】** 木村 秀二**【電話番号】** 03-5276-3241**【手数料の表示】****【予納台帳番号】** 003458**【納付金額】** 21,000円**【提出物件の目録】****【物件名】** 明細書 1**【物件名】** 図面 1**【物件名】** 要約書 1**【包括委任状番号】** 0102485**【プルーフの要否】** 要

【書類名】 明細書

【発明の名称】 ログ取得方法およびプログラム、記憶媒体

【特許請求の範囲】

【請求項 1】 所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、ポインタ・パラメータが所定の定義方法で定義されているか否かを判断する工程と、

ポインタ・パラメータが所定の定義方法で定義されていた場合、該定義方法に基づいて該ポインタ・パラメータの指すメモリの内容をログとして記録する工程と

を備えることを特徴とするログ取得方法。

【請求項 2】 前記ログ取得のための関数は、前記定義方法に基づいてメモリサイズを算出する工程を更に備え、前記記録する工程は、前記ポインタ・パラメータの指すメモリの内容を、該算出したメモリサイズ分、記録することを特徴とする請求項 1 に記載のログ取得方法。

【請求項 3】 所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行う関数を呼び出す際の所定の情報と、前記実行結果を受け取った際の所定の情報とを記録する工程と、

前記プログラム内の関数定義に基づいて、エクスポートされていない関数の有無を判断する工程と、

エクスポートされていない関数があった場合、該関数に対応するログ取得のための関数を新たに生成し、該エクスポートされていない関数のアドレスを、該新たに生成された関数のアドレスに書き換える工程と、を備え、

前記新たに生成された関数は、

前記エクスポートされていない関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記エクスポートされていない関数を呼び出す際の所定の情報と、前記実行結果を受け取った際の所定の情報とを記録する工程と

を備えることを特徴とするログ取得方法。

【請求項 4】 所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、可変長配列としてポインタ・パラメータが定義されているか否かを判断する工程と、

可変長配列としてポインタ・パラメータが定義されていた場合、該ポインタ・パラメータの指すメモリの内容を該指定された配列で記録する工程と

を備えることを特徴とするログ取得方法。

【請求項 5】 所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った

実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義に、構造体としてポインタ・パラメータが指定されているか否かを判断する工程と、

構造体としてポインタ・パラメータが指定されていた場合、該ポインタ・パラメータの指すメモリの内容について、前記プログラム内の関数定義において定義された構造体のサイズ分ログとして記録する工程と

を備えることを特徴とするログ取得方法。

【請求項 6】 所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、構造体として種類指定がなされたポインタ・パラメータが定義されているか否かを判断する工程と、

構造体として種類指定がなされたポインタ・パラメータが定義されていた場合、該ポインタ・パラメータの指すメモリの内容を、種類指定がなされたデータ型として記録する工程と

を備えることを特徴とするログ取得方法。

【請求項 7】 所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、ポインタ・パラメータが所定の定義方法で定義されているか否かを判断する工程と、

ポインタ・パラメータが所定の定義方法で定義されていた場合、該定義方法に基づいて該ポインタ・パラメータの指すメモリの内容をログとして記録する工程と

を備えることを特徴とするログ取得方法。

【請求項 8】 前記ログ取得のためのメソッドは、前記定義方法に基づいてメモリサイズを算出する工程を更に備え、前記記録する工程は、前記ポインタ・パラメータの指すメモリの内容を、該算出したメモリサイズ分、記録することを特徴とする請求項 7 に記載のログ取得方法。

【請求項 9】 所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行うメソッドを呼び出す際の所定の情報と、前記実行結果を受け取った際の所定の情報とを記録する工程と、

前記プログラム内の関数定義に基づいて、エクスポートされていないメソッドの有無を判断する工程と、

エクスポートされていないメソッドがあった場合、該メソッドに対応するログ取得のためのメソッドを新たに生成し、該エクスポートされていないメソッドのアドレスを、該新たに生成されたメソッドのアドレスに書き換える工程と、を備え、

前記新たに生成されたメソッドは、

前記エクスポートされていないメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記エクスポートされていないメソッドを呼び出す際の所定の情報と、前記実行結果を受け取った際の所定の情報とを記録する工程と

を備えることを特徴とするログ取得方法。

【請求項 10】 所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、可変長配列としてポインタ・パラメータが定義されているか否かを判断する工程と、

可変長配列としてポインタ・パラメータが定義されていた場合、該ポインタ・パラメータの指すメモリの内容を該指定された配列で記録する工程と

を備えることを特徴とするログ取得方法。

【請求項 11】 所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義に、構造体としてポインタ・パラメータが指定されているか否かを判断する工程と、

構造体としてポインタ・パラメータが指定されていた場合、該ポインタ・パラメータの指すメモリの内容について、前記プログラム内の関数定義において定義された構造体のサイズ分ログとして記録する工程と

を備えることを特徴とするログ取得方法。

【請求項 12】 所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、構造体として種類指定がなされたポインタ・パラメータが定義されているか否かを判断する工程と、

構造体として種類指定がなされたポインタ・パラメータが定義されていた場合、該ポインタ・パラメータの指すメモリの内容を、種類指定がなされたデータ型として記録する工程と

を備えることを特徴とするログ取得方法。

【請求項 13】 請求項 1 乃至 12 のいずれか 1 つに記載のログ取得方法をコンピュータによって実現させるための制御プログラムを格納した記憶媒体。

【請求項 14】 請求項 1 乃至 12 のいずれか 1 つに記載のログ取得方法をコンピュータによって実現させるための制御プログラム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、複数にモジュール分けされたソフトウェアの処理ログを取得するための技術に関するものである。

【0002】

【従来の技術】

従来より、再現率の低いソフトウェアの障害に対しては、ソフトウェアの処理ログを取得し、当該処理ログを解析することによって、障害の原因をつきとめ、その対策を講じてきた。

【0003】

【発明が解決しようとする課題】

しかし、上記従来の処理ログの取得には以下のような問題点がある。

(1) ユーザの動作環境でもログを取得しつづけるためには、ソフトウェアのモジュール自体に手を加え、処理ログ取得ルーチンを追加しなければならず、処理ログ取得のための作業負荷が大きかった。

(2) 処理ログ取得はモジュール毎に行うため、生成されたログはモジュール単位のものとなってしまい、ソフトウェア全体の処理を、完全に時間順のログとして取得するのが困難である。このため、全体の処理ログとしての見通しが悪く、ログを解析して障害の原因を発見するまでのプロセスに工数がかかっていた。

【0004】

本発明は、上記課題を鑑みてなされたものであり、複数のモジュール分けされたソフトウェアの処理ログを容易に取得でき、かつ、ソフトウェアの障害の原因の解析のための工数を削減することが可能なログ取得方法、ならびに該方法をコンピュータによって実現させるためのプログラム、および該プログラムを格納した記憶媒体を提供することを目的とする。

【0005】

【課題を解決するための手段】

上記の目的を達成するために本発明に係るログ取得方法は以下のような構成を備える。即ち、

所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、ポインタ・パラメータが所定の定義方法で定義されているか否かを判断する工程と、

ポインタ・パラメータが所定の定義方法で定義されていた場合、該定義方法に基づいて該ポインタ・パラメータの指すメモリの内容をログとして記録する工程とを備える。

【0006】

【発明の実施の形態】

【第1の実施形態】

本実施形態は、あるモジュールから別のモジュール内に存在する関数の呼び出しが行われる際の仕組みである、メモリに保持されたインポート関数アドレス、または仮想関数アドレステーブル (Virtual Address Table) を利用して、モジュール間の関数呼び出しをフックしてログに記録することで、ソフトウェアのモジュール自体に手を加えることなく、ソフトウェア全体の処理を、時間順のログとして取得することを可能にするものである。以下に具体的に説明する。

【0007】

<システム構成>

図1は、本発明の各実施形態にかかるログ取得方法を実現するコンピュータ（ソフトウェア評価システム）の構成をあらわす図である。説明を簡略化するために、本実施形態では、本ソフトウェア評価システムが1台のPC内部に構築されるものとするが、本発明のログ取得方法の特徴は1台のPC内部に構築されるか、あるいは複数のPCにネットワークシステムとして構築されるかによらず有効である。

【0008】

本ソフトウェア評価システムを搭載するコンピュータには、CPU1、チップセット2、RAM3、ハードディスクコントローラ4、ディスプレイコントローラ5、ハードディスクドライブ6、CD-ROMドライブ7、ディスプレイ8が搭載されている。また、CPUとチップセットを繋ぐ信号線11、チップセット2とRAM3とを繋ぐ信号線12、チップセット2と各種周辺機器とを繋ぐ周辺機器バス13、ハードディスクコントローラ4とハードディスクドライブ6とを繋ぐ信号線14、ハードディスクコントローラ4とCD-ROMドライブ7とを繋ぐ信号線15、ディスプレイコントローラ5とディスプレイ8とを繋ぐ信号線16が搭載されている。

【0009】

<関数処理に対する処理ログ取得>

本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムを説明するために、まず図2によって、複数のモジュールに分かれたソフ

トウェアが、通常の状態でどのようにメモリにロードされるかを説明する。

【0010】

通常、複数のモジュールに分かれたソフトウェアは、全体の制御を行う実行ファイル EXE(23)と、モジュールとして存在し EXE の補完的な役割を担うダイナミックリンクライブラリ DLL(27)とに分かれており、メモリには EXE と DLL の両方がロードされる。EXE はコードセグメント(28)とデータセグメント(29)、そしてインポート関数アドレステーブル(22)から成っている。更に、インポート関数アドレステーブルは、関数の所属する DLL によって分かれており(21, 24)、DLL ごとにそれぞれの関数がロードされたアドレスが書かれている(30~35)。DLL の関数の実体は、DLL ごとに分かれて(25, 26)ロードされ、それぞれの関数は該当する DLL の一部としてロードされる(36~41)。この図では、1本の EXE が A.DLL 及び B.DLL の2つのダイナミックリンクライブラリ内の関数を使用している例を示しており、実際に使用される関数は Func AA, Func AB, Func AC, Func BA, Func BB, Func BC の6個となっている。

【0011】

EXE のコードセグメント内にあるコードが関数 Func AA を呼び出す場合には、まずインポート関数アドレステーブル内に書かれた Func AA のアドレス(30)が読み込まれる。ここには実際には A.DLL の一部として読み込まれた Func AA コード(36)のアドレスが書かれており、そのアドレスをコールすることによって、EXE のコードは A.DLL の Func AA を呼び出すことができる。

【0012】

図3は、本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図であり、図2とは、ログ取得用のコードに対して IAT Patch(Import Address Table Patch)という手法を用いて、関数呼び出しをリダイレクトしているという点で異なっている。

【0013】

ログ取得が開始されると、メモリ内にはIAT Patch用のDLLであるC.DLL(58)がロードされる。C.DLLはインポート関数アドレステーブル(52)内に書かれた関数のアドレスを、C.DLL内のログ取得コードであるFunc CAA, Func CAB, Func CAC, Func CBA, Func CBB, Func CBCのアドレスに書き換える(61~66)。C.DLL内のFunc CAA, Func CAB, Func CAC, Func CBA, Func CBB, Func CBCのコード(73~78)は、ログを記録すると共に、元々の関数呼び出しを受けるべくメモリにロードされている、該当する関数であるFunc AA, Func AB, Func AC, Func BA, Func BB, Func BC(67~72)を呼び出す。

【0014】

図4Aは、図3におけるIAT Patchの処理をあらわす図、図4Bはログ取得処理の流れを示すフローチャートである。説明を簡略化するために、この図ではEXEがA.DLL内のFunc AAを呼び出す際に、IAT Patchによるログ取得コードがどのように動作するか例をあらわしている。

【0015】

EXE(図4Aの91)がFunc AAをコールすると(図4Aの94)、C.DLL内にあるログ取得コードがDLL名/関数名をメモリに保存し(図4BのステップS402)、呼び出し時刻をメモリに保存し、呼び出し時のパラメータをメモリに保存し、呼び出し時のポインタパラメータの指すメモリ内容を、別メモリに保存する(図4Aの95、図4BのステップS403)。その後C.DLLは本来呼び出されるはずであった、A.DLL(図4Aの93)内のFunc AAをコールする(図4Aの96、図4BのステップS404)。A.DLLのFunc AA処理(図4Aの97)が終了し、C.DLLに制御がリターンすると(図4Aの98)、C.DLLはリターン時の時刻をメモリに保存し、戻り値をメモリに保存し、リターン時にポインタパラメータが指すメモリ内容を、別メモリに保存する(図4Aの99)。その後、C.DLLは保存したログ情報をファイルに書き込み(図4Aの100、図4BのステップS405)、あたかもA.DLLのF

unc AAが通常通りに終了したかのように、EXEにリターンする(101)。
。

【0016】

図5は、本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの内部構成をあらわす図である。通常は実行形式のEXE(113)が、DLL-1(116)やDLL-2(117)内の関数を呼び出すが、ここではAPIトレーサと呼ばれるログ取得コードを埋め込み(114)、処理ログを生成している(115)。APIトレーサは、DLL-1やDLL-2の関数定義を記述したファイル(111)と、どのDLLのどの関数のインポート関数テーブルを書き換えてログを取得するかの設定シナリオ(トレースシナリオ112)を元に動作する。

【0017】

＜メソッド処理に対する処理ログ取得＞

次に、本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、実行ファイルEXE(118)がCOM(Component Object Model)サーバでエクスポートされているインターフェースのインスタンス作成時に、どのようにメモリにロードされるかを説明するために、まず、図6によって、通常の状態でどのようにメモリにロードされるかを説明する。

【0018】

通常、インターフェースのインスタンス作成を行うと、COMサーバ内で、要求されたインターフェース(121, 122)と、そのメソッド(:オブジェクト指向プログラミングにおいて、オブジェクトの実行する手続きを記述したプログラム、130~135)が作成され、それらは、メモリ上に両方がロードされる。ここで、virtual address table(仮想アドレステーブル)は作成された各インターフェース毎に作られ(118, 120)、作成要求を行ったEXEに渡される。このvirtual address tableには各メソッドの作成されたアドレスが書かれている(124~129)。EXEはこれら情報を利用し、各インターフェースに対して呼び出しを行う。この図では

、1本のEXEがInterface A及びInterface Bの2つのインターフェースのインスタンスを作成しており、そのインターフェース内部のメソッドを使用している例を示しており、実際に使用されているメソッドは、Method AA, Method AB, Method AC, Method BA, Method BB, Method BCとなっている。

【0019】

EXEのコードが関数Method AAを呼び出す場合には、まずvirtual address table内に書かれたMethod AAのアドレス(124)が読み込まれる。ここには実際にはCOMサーバのInterface Aの一部として作成されたMethod AAコード(130)のアドレスが書かれており、そのアドレスをコールすることによって、EXEのコードはInterface AのMethod AAを呼び出すことができる。

【0020】

図7は、本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図であり、図6とは、ログ取得用のコードに対してVTable Patch(virtual address table Patch)という手法を用いて、メソッド呼び出しをリダイレクトしているという点で異なっている。

【0021】

ログ取得が開始されると、メモリ内にはVTable Patch用のDLL(143)がロードされる。このDLLはvirtual address table(136, 138)内に書かれたメソッドのアドレスを、DLL内のログ取得コードであるMethod A'A, Method A'B, Method A'C, Method B'A, Method B'B, Method B'Cのアドレスに書き換える(145~150)。DLL内のMethod A'A, Method A'B, Method A'C, Method B'A, Method B'B, Method B'Cのコード(157~162)は、ログを記録すると共に、元々のメソッド呼び出しを受けるべくメモリにロードされている、該当するメソッドであるMethod AA, Method AB, Me

t h o d A C, M e t h o d B A, M e t h o d B B, M e t h o d B C (157~162)を呼び出す。

【0022】

図8Aは、図7におけるV T a b l e P a t c hの処理をあらわす図、図8Bはログ取得処理の流れを示すフローチャートである。説明を簡略化するために、この図ではE X EがC O Mサーバ内のI n t e r f a c e AのM e t h o d A Aを呼び出す際に、V T a b l e P a t c hによるログ取得コードがどのように動作するか例をあらわしている。

【0023】

E X E (図8Aの163)がM e t h o d A Aをコールすると(図8Aの166)、D L L内にあるログ取得コードがモジュール名/インターフェース名/メソッド名をメモリに保存し(図8BのステップS802)、呼び出し時刻をメモリに保存し、呼び出し時のパラメータをメモリに保存し、呼び出し時のポインタパラメータの指すメモリ内容を、別メモリに保存する(図8Aの167、図8BのステップS803)。その後D L Lは本来呼び出されるはずであった、C O Mサーバ(図8Aの165)内のM e t h o d A Aをコールする(図8Aの168、図8BのステップS804)。C O MサーバのM e t h o d A A処理(図8Aの169)が終了し、D L Lに制御がリターンすると(図8Aの170)、D L Lはリターン時の時刻をメモリに保存し、戻り値をメモリに保存し、リターン時にポインタパラメータが指すメモリ内容を、別メモリに保存する(図8Aの171)。その後、D L Lは保存したログ情報をファイルに書き込み(図8Aの172、図8BのステップS805)、あたかもC O MサーバのM e t h o d A Aが通常通りに終了したかのように、E X Eにリターンする(図8Aの173)。

【0024】

図9は、本発明の第1の実施形態にかかるソフトウェア評価システムの内部構成をあらわす図である。通常は実行形式のE X E (176)が、C O Mサーバ1 (179)やC O Mサーバ2 (180)内のメソッドを呼び出すが、ここではA P Iトレーサと呼ばれるログ取得コードを埋め込み(177)、処理ログを生成している(178)。A P Iトレーサは、C O Mサーバ1 (179)やC O Mサーバ2の関

数定義を記述したファイル(174)と、どのCOMサーバのどのインターフェースのどのメソッドの `virtual address table` を書き換えてログを取得するかの設定シナリオ(175)を元に動作する。

【0025】

以上の説明から明らかなように、本実施形態にかかるログ取得方法によれば、複数にモジュール分けされたソフトウェアの処理ログの取得において、モジュール自体に変更を加えることなく、モジュール内に用意された関数／メソッドの呼び出しをログとして記録することが可能となり、処理ログ取得のための作業負荷を低減することが可能となる。また、生成されるログは、時間順のログとして取得することができ、ログの解析が容易となるため、ソフトウェアの障害の原因の解析のための工数を削減することが可能となる。

【0026】

【第2の実施形態】

本実施形態では、通常では、取得しきれないポインタ・パラメータのデータをバイナリとしてログ取得する場合について述べる。

【0027】

図10は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義の一例であり、一般的に広く用いられているIDLにより、記述されている。本実施形態にかかるログ取得方法を実現するソフトウェア評価システムに於いては、このIDLをトークン化したタイプライブラリファイルを関数定義ファイルとして使用する。

【0028】

図11は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いて、ポインタ・パラメータに対してバイナリ取得を指定することで、ポインタ・パラメータのデータの実体をログとして取得する為のIDLによる記述である。

【0029】

FuncBinId関数の定義に於いて、`long* lpParam`に対して`custum(PAT_PARAM_ATTR_ID, "binid_is()")`と宣言している(201)。ここで、`PAT_PARAM_ATTR_ID`(

2 0 0)はIDLの中で本ソフトウェア評価システムが利用するための識別子である。ここで、"binid_is()"と定義することにより、このパラメータが指すポインタからパラメータのデータ型サイズ(ここではlong型)分のデータを、バイナリデータとしてログに保存する。

【 0 0 3 0 】

FuncSizeIs関数の定義に於いて、int* lpParamに対してcustum(PAT_PARAM_ATTR_ID, "size_is(dwCount)")と宣言している(2 0 2)。ここで、"size_is(dwCount)"と定義することにより、このパラメータは第一パラメータであるdwCount個分のデータが有ると扱われる。そして、このパラメータが指すポインタから、(dwCount×パラメータのデータ型サイズ(ここでは、int型))分のデータをバイナリデータとしてログに保存する。

【 0 0 3 1 】

FuncLengthIs関数の定義に於いて、char* lpszParamに対してcustum(PAT_PARAM_ATTR_ID, "length_is(dwLength)")と宣言している(2 0 3)。ここで、"length_is(dwLength)"と定義することにより、このパラメータは第一パラメータであるdwLength個分のデータが有ると扱われる。そして、このパラメータが指すポインタから、(dwLength×パラメータのデータ型サイズ(ここではchar型))分のデータをバイナリデータとしてログに保存する。

【 0 0 3 2 】

FuncBytesIs関数の定義に於いて、void* lpParamに対してcustum(PAT_PARAM_ATTR_ID, "bytes_is(dwSize)")と宣言している(2 0 4)。ここで、"bytes_is(dwSize)"と定義することにより、このパラメータは第一パラメータであるdwSizeバイト分のデータが有ると扱われる。そして、このパラメータが指すポインタから、dwSizeバイト分のデータをバイナリデータとしてログに保存する。

【 0 0 3 3 】

FuncBytesIsl関数の定義に於いて、void* lpParamに対してcustum(PAT_PARAM_ATTR_ID, "bytes_is(12)")と宣言している(2 0 5)。ここで、"bytes_is(12)"と定義することにより、このパラメータは指定された12バイト分のデータが有ると扱われる。そして、このパラメータが指すポインタから、12バイト分のデー

タをバイナリデータとしてログに保存する。

【0034】

図12は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図11の様に関数定義されている時のログを取得する場合の流れを示す図である。

【0035】

処理が開始されると(ステップS1)、ログ取得を開始し、モジュール名、インターフェース名、関数/メソッド名をHDDに保存する(ステップS2)。次にログ取得コードは、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をHDDに保存する(ステップS3)。そして、関数定義にバイナリ取得の設定が存在するか判別し(ステップS4)、存在する場合には、保存するメモリサイズを各定義方法(binid_is, size_is, length_is, bytes_is)毎に計算し(ステップS5)、ポインタ・パラメータの指すメモリの内容を算出したサイズ分HDDに保存する(ステップS6)。続いて、元の関数を呼び出す(ステップS7)。関数からリターンすると、ログ取得コードはリターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をHDDに保存する(ステップS8)。そして、関数定義にバイナリ取得の設定が存在するか判別し(ステップS9)、存在する場合には、保存するメモリサイズを各定義方法(binid_is, size_is, length_is, bytes_is)毎に計算し(ステップS10)、ポインタ・パラメータの指すメモリの内容を算出したサイズ分HDDに保存する(ステップS11)。この処理はユーザから終了の命令があると(ステップS12)終了される。

【0036】

図13は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図11の定義により取得されたログデータの図である。

【0037】

関数定義により、バイナリ取得の設定がない場合に取得できるログ(210)は、DataIDを除いた部分である。そして、バイナリ取得の設定を行った場合には、DataID及び、バイナリデータのログ(211)が取得できる。例えばFuncSizeIsのログを比較すると、バイナリ取得の設定がない場合には、ポインタ・パラメータ

lpParamが指し示すデータをint型で1個分だけは取得可能である。しかし、実際にはこのデータは10個のint型データの配列を指し示す。そのためバイナリ取得の設定を行った場合には、int型データのサイズである4バイト×10個分、計40バイトのデータを取得可能となる。

【0038】

以上の説明から明らかなように、本実施形態によれば、サイズとポインタ・パラメータを関連付けることにより、通常では、取得しきれないポインタ・パラメータのデータをバイナリとしてログ取得可能となる。

【0039】

【第3の実施形態】

本実施形態では、コールバック関数等のエクスポートされていない関数をログとして取得する場合について述べる。

【0040】

図14は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いてコールバック関数等のエクスポートされていない関数をログとして取得するためのIDLによる記述である。

【0041】

FuncSetCallback関数の定義に於いて、DWORD pfnFuncCallbackに対してcustom(PAT_PARAM_ATTR_ID, "funcname_is(FuncCallback)")と宣言している(221)。FuncSetCallback関数は、モジュールに対してコールバック関数を設定する関数であり、DWORD pfnFuncCallbackは、そのコールバック関数のアドレスを設定するためのパラメータである。ここで、"funcname_is(FuncCallback)"と定義することにより、ログ取得処理は、このパラメータに渡された値をFuncCallback関数(222)のアドレスとして認識し、ログ取得処理のアドレスに置換する。また、ログ取得処理時に本来のコールバック関数を呼ぶためにオリジナルの値を保存する。これにより、エクスポートされていないコールバック関数のログが取得可能となる。

【0042】

GetFuncPointer関数の定義に於いて、DWORD pfnFuncInternalに対してcustom(

PAT_PARAM_ATTR_ID, "funcname_is(FuncInternal)")と宣言している(2 2 3)。GetFuncPointer関数は、モジュール内部のエクスポートされていない関数のポインタを取得する関数であり、DWORD pfnFuncInternalは、その非エクスポート関数のポインタを取得するためのパラメータである。ここで、"funcname_is(FuncInternal)"と定義することにより、ログ取得処理は、このパラメータに渡された値をFuncInternal関数(2 2 4)のアドレスとして認識し、ログ取得処理のアドレスに置換する。また、ログ取得処理時に本来の非エクスポート関数を呼ぶためにオリジナルの値を保存する。これにより、エクスポートされていないモジュール内部の関数のログが取得可能となる。

【0 0 4 3】

GetFuncPointeArray関数(2 2 5)は、モジュール内部のエクスポートされていない関数群のアドレスをFUNCPOINTERARRAY構造体(2 2 0)に取得するための関数である。ここで、FUNCPOINTERARRAYの各メンバの定義に対してcustom(PAT_PARAM_ATTR_ID, "funcname_is(FuncInternal1~4)")と宣言している。ここで、"funcname_is(FuncInternal1~4)"と定義することにより、この構造体のメンバに渡された値をFuncInternal1~4関数(2 2 6)のアドレスとして認識し、ログ取得処理のアドレスに置換する。また、ログ取得処理時に本来の非エクスポート関数を呼ぶためにオリジナルの値を保存する。これにより、エクスポートされていないモジュール内部の関数群のログを取得可能となる。

【0 0 4 4】

図 1 5 は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図であり、図 3 とは、隠し関数のログ取得用のコードが追加されている点で異なっている。A.DLLにある、FuncAD(2 3 1)は、エクスポートされていない関数であり、そのため、Import Address Table(2 3 0)にも、存在しない。この時funcname_isをFuncADに対して定義すると、C.DLLにログ取得のためのFuncCAD(2 3 2)が生成され、本来FuncADのアドレスが返されるfuncname_isで定義されたパラメータの値を置換する場合に、FuncCADのアドレスが使用される。

【0 0 4 5】

図16は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図14の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【0046】

処理が開始されると(ステップS21)、ログ取得を開始し、モジュール名、インターフェース名、関数／メソッド名をHDDに保存する(ステップS22)。次にログ取得コードは、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をHDDに保存する(ステップS23)。そして、関数定義にfuncname_isの設定が存在するか判別し(ステップS24)、存在する場合には、funcname_isによって定義されている関数定義を関数定義ファイルから取得し、その定義に基づいてログ取得処理用のコードを生成する(ステップS25)。そして、funcname_isが定義されている値を保存し、生成したログ取得コードのアドレスに置換する(ステップS26)。

【0047】

続いて、元の関数を呼び出す(ステップS27)。関数からリターンすると、ログ取得コードはリターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をHDDに保存する(ステップS28)。そして、関数定義にfuncname_isの設定が存在するか判別し(ステップS29)、存在する場合には、funcname_isによって定義されている関数の定義を関数定義ファイルから取得し、その定義に基づいてログ取得処理用のコードを生成する(ステップS30)。そして、funcname_isが定義されている値を保存し、生成したログ取得コードのアドレスに置換する(ステップS31)。この処理はユーザから終了の命令があると(ステップS32)終了される。

【0048】

funcname_isで設定されているパラメータが生成されたログ取得コードのアドレスに置換されると、その後そのアドレスを使用して呼び出される元の関数は、通常のログと同様に処理されるようになる(つまり、生成されたログ取得コードは、エクスポートされていない関数を呼び出し、処理を実行させ、受け取った実行結果を渡すとともに、エクスポートされていない関数を呼び出す際の所定の情

報と、実行結果を受け取った際の所定の情報とをログとして記録する)。

【0 0 4 9】

図 1 7 は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 1 4 の定義を行わない場合に取得されたログデータの図である。関数定義ファイルに於いて設定を行わない場合は、コールバック関数、エクスポートされていない内部関数のログは取得できない。

【0 0 5 0】

図 1 8 は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 1 4 の定義より取得されたログデータの図である。関数定義ファイルに於いて、コールバック関数、エクスポートされていない内部関数を取得するように設定しているので、FuncCallBack関数や、FuncInternal関数、FuncInternal4関数等の非エクスポート関数のログが取得可能となる。

【0 0 5 1】

このように本実施形態によれば、通常の方法では取得できない非エクスポート関数のログを取得可能になるという効果が得られる。

【0 0 5 2】

【第 4 の実施形態】

本実施形態では、通常の方法では取得できない関数のログとして、可変長配列のパラメータのログを取得する場合について述べる。

【0 0 5 3】

図 1 9 は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いて可変長配列のパラメータをログとして取得するための I D L による記述である。

【0 0 5 4】

FuncArrayIs関数の定義に於いて、int* lpnParamに対してcustum(PAT_PARAM_ATTR_ID, "array_is(dwCount)")と宣言している(2 4 0)。ここで、" array_is(dwCount)"と定義することにより、このポインタ・パラメータは第一パラメータであるdwCount個分のint型配列であると扱われる。そして、このポインタ・パラメータが指すポインタをint型のdwCount個の配列とし、データをログに保存する。

【0055】

図20は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図19の様に関数が定義されている時のログを取得する場合の流れ図である。

【0056】

処理が開始されると(ステップS41)、ログ取得を開始し、モジュール名、インターフェース名、関数/メソッド名をHDDに保存する(ステップS42)。次に関数定義に可変長配列取得(array_is)の設定が存在するか判別し(ステップS43)、存在する場合には、ポインタ・パラメータで定義されているパラメータを配列の定義として扱い(ステップS44)、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をHDDに保存する(ステップS45)。そして、続いて、元の関数を呼び出す(ステップS46)。関数からリターンすると、ログ取得コードは関数定義に可変長配列取得(array_is)の設定が存在するか判別し(S47)、存在する場合には、ポインタ・パラメータで定義されているパラメータを配列の定義として扱い(ステップS48)、リターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をHDDに保存する(ステップS49)。この処理はユーザから終了の命令があると(ステップS50)終了される。

【0057】

図21は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図19の定義を行わない場合に取得されたログデータ(250)と、図19の定義により取得されたログデータ(251)の図である。関数定義ファイルに於いて設定を行わない場合は、ポインタ・パラメータは配列としては扱えないので、先頭のデータのみが取得されているが、関数定義ファイルに於いて設定が行われている場合では、ポインタ・パラメータは配列として扱われるため、配列の全データが取得されている。

【0058】

このように本実施形態によれば、通常の方法では取得できない関数のログとして、可変長配列のパラメータのログが取得可能になるという効果が得られる。

【0059】

【第5の実施形態】

本実施形態では、通常の間数定義では、パラメータを取得できない関数について、ログを取得する場合について説明する。

【0060】

図22は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、通常の間数定義では、パラメータを取得できない関数の例である。

【0061】

構造体として、STRUCTSIZE1、STRUCTSIZE2、STRUCTSIZE3の3種類が定義されており、各構造体のDWORD dwSizeメンバには、自分自身の構造体のサイズを入れる必要がある。FuncGetData関数は、第一パラメータdwKindと、第二パラメータlpBufに上記3種類のどの構造体のポインタが渡されているかを設定する。FuncGetData関数は、第一パラメータが1の場合には、lpBufをSTRUCTSIZE1のポインタとして扱って処理を行い、第一パラメータが2の場合には、lpBufをSTRUCTSIZE2のポインタとして扱って処理を行い、第一パラメータが3の場合には、lpBufをSTRUCTSIZE3のポインタとして扱って処理を行う。この場合、通常の間数定義により、FuncGetDataを定義すると、lpBufはvoid型のポインタとなり、データを取得することはできない。

【0062】

図23は、図22に於いて、STRUCTSIZE1、STRUCTSIZE2、STRUCTSIZE3の各構造体がどのようにメモリを使用するかのメモリ図である。構造体STRUCTSIZE1(260)の、各メンバは、オフセット0x000にDWORD dwSize(261)、0x0004にDWORD dwParam1(262)、0x0008にDWORD dwParam2、0x000CにDWORD dwParam3が存在する。構造体STRUCTSIZE2(265)はDWORD dwSize～DWORD dwParam3(266～269)までは、STRUCTSIZE1と同様であり、0x0010にDWORD dwParam4(270)が存在する。STRUCTSIZE3(271)はDWORD dwSize～DWORD dwParam4(272～276)までは、STRUCTSIZE2と同様であり、0x0014にDWORD dwParam5(277)が存在する。このように、STRUCTSIZE3のdwSize～dwParam4までのメモリ配置は、STRUCTSIZE2と同様であり、STRUCTSIZE2のdwSize～dwP

aram3までのメモリ配置は、STRUCTSIZE1と同様になっている。

【0063】

図24は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図22の様な関数のパラメータのログを取得するためのIDLによる記述である。

【0064】

本来の関数では、STRUCTSIZE1、STRUCTSIZE2、STRUCTSIZE3と分かれていた構造体をSTRUCTSIZEという1個の構造体として定義する(291)。この定義はSTRUCTSIZE3と同様であり、これは、図23でのメモリ配置から、STRUCTSIZE3は、STRUCTSIZE1、STRUCTSIZE2と同様のメモリ配置部分を持つからである。そして、この構造体のメンバであり、構造体のサイズを表す、DWORD dwSizeに対して、[custom(PAT_PARAM_ID, "structsize_is()")]を設定する。この事で、ログ取得処理は構造体のデータ解析を行う場合に、構造体のサイズを知ることが可能となる。また、FuncGetData関数において、第二パラメータのlpBufをSTRUCTSIZE型として定義しておく。こうすることで、このパラメータをログとして保存する場合には、STRUCTSIZE型として処理が行われる。仮に、本手法を用いず、STRUCTSIZE3のデータを取得するように関数定義ファイルに記述した場合には、GetFuncDataがdwKind=1や2で呼ばれた場合には、STRUCTSIZE1やSTRUCTSIZE2のデータが存在することになり、dwParam3やdwParam4のデータをログに取得しようとしてしまうために、メモリ例外が発生する。

【0065】

図25は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【0066】

処理が開始されると(ステップS61)、ログ取得を開始し、モジュール名、インターフェース名、関数/メソッド名をHDDに保存する(ステップS62)。次に関数定義に構造体のサイズ指定(structsize_is)の設定が存在するか判別し(ステップS63)、存在する場合には、設定されている構造体パラメータのデータ

をサイズ指定(structsize_is)の設定で定義されているデータサイズ範囲内で解析する(ステップS64)。そして、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をHDDに保存する(ステップS65)。そして、続いて、元の関数を呼び出す(ステップS66)。関数からリターンすると、関数定義に構造体のサイズ指定(structsize_is)の設定が存在するか判別し(ステップS67)、存在する場合には、設定されている構造体パラメータのデータをサイズ指定(structsize_is)の設定で定義されているデータサイズ範囲内で解析する(ステップS68)。そして、リターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をHDDに保存する(ステップS69)。この処理はユーザから終了の命令があると(ステップS70)終了される。

【0067】

図26は、図25で行われる構造体パラメータ解析の詳細をメモリ配置を基に表した図である。

【0068】

実際のプログラムでGetFuncDataがdwKind=1で使用された場合には、STRUCTSIZE構造体(300)が使用され、その時の各メンバ(301～304)のメモリ使用は、図のようになっている。図24の様に関数定義がなされている場合には、本ソフトウェア評価システムでは、STRUCTSIZE構造体(305)として認識され、その時の各メンバ(306～311)のメモリ使用は図のようになると推測する。そして、サイズ指定(structsize_is)の設定が存在する場合には、dwSizeの値を調べることで、構造体のサイズがdwSizeバイトであると知ることが可能となり、STRUCTSIZE構造体のうち、dwSizeバイト分(312)のデータのみをログとして取得する。

【0069】

図27は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図24の定義により取得されたログデータの図である。通常関数定義では、void*となり、ポインタのみしか取得できない構造体のデータが、使用される構造体の種類により、その構造体のデータがログとして取得されている。

【 0 0 7 0 】

このように、本実施形態によれば、通常の方法では取得できないパラメータのログを取得可能になるという効果が得られる。

【 0 0 7 1 】**【第 6 の実施形態】**

図 2 8 は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、通常関数定義では、パラメータを取得できない関数の例である。

【 0 0 7 2 】

構造体として、STRUCTKIND1, STRUCTKIND2, STRUCTKIND3の3種類が定義されている。GetFuncData関数は、第一パラメータdwKindに第二パラメータlpBufに上記3種類のどの構造体のポインタが渡されているかを設定する。FuncGetData関数は、第一パラメータが1の場合には、lpBufをSTRUCTKIND1のポインタとして扱って処理を行い、第一パラメータが2の場合には、lpBufをSTRUCTKIND2のポインタとして扱って処理を行い、第一パラメータが3の場合には、lpBufをSTRUCTKIND3のポインタとして扱って処理を行う。この場合、通常関数定義により、FuncGetDataを定義すると、lpBufはvoid型のポインタとなり、データを取得することはできない。

【 0 0 7 3 】

図 2 9 は、図 2 8 に於いて、STRUCTKIND1, STRUCTKIND2, STRUCTKIND3の各構造体がどのようなメモリ配置になるかを示した図である。構造体STRUCTKIND1(3 3 0)の、各メンバは、オフセット0x0 0 0 0にchar chParam(3 3 1)、0x0 0 0 1にDWORD dwParam(3 3 2)、0x0 0 0 5にshort shParam(3 3 3)が存在する。構造体STRUCTKIND2(3 3 4)の、各メンバは、オフセット0x0 0 0 0にshort shParam(3 3 5)、0x0 0 0 2にDWORD dwParam(3 3 6)、0x0 0 0 6にchar chParam(3 3 7)が存在する。構造体STRUCTKIND3(3 3 8)の、各メンバは、オフセット0x0 0 0 0にchar chParam(3 3 9)、0x0 0 0 1にshort shParam(3 4 0)、0x0 0 0 3にDWORD dwParam(3 4 1)、0x0 0 0 7にlong lParam(3 4 2)、0x0 0 0 Bにint nParamが存在する。このように、各構造体にサイズの情報が無く、構造体データのメモリ構造も異なるため、第5の実施形態で示し

た方法は使用できない。

【 0 0 7 4 】

図 3 0 は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図 2 8 の様な関数のパラメータのログを取得するための I D L による記述である。

【 0 0 7 5 】

各構造体は、通常の方法で定義されている。FuncGetData関数の第二パラメータ void* lpBuf に対して [custum(PAT_PARAM_ID, "structkind_is(dwKind:1:STRUCTKIND1*, 2:STRUCTKIND2*, 3:STRUCTKIND3*))"] と設定している。これにより lpBuf のデータ型は、第一パラメータ dwKind の値が 1 の場合には、STRUCTKIND1* として、2 の場合には、STRUCTKIND2* として、3 の場合には、STRUCTKIND3* として扱われ、ログとして保存される。

【 0 0 7 6 】

図 3 1 は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図 3 0 の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【 0 0 7 7 】

処理が開始されると(ステップ S 8 1)、ログ取得を開始し、モジュール名、インターフェース名、関数／メソッド名をメモリに保存する(ステップ S 8 2)。次に関数定義に構造体の種類指定(structkind_is)の設定が存在するか判別し(ステップ S 8 3)、存在する場合には、設定されている構造体パラメータのデータを種類指定(structkind_is)の設定で定義されているデータ型として解析する(ステップ S 8 4)。そして、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をメモリに保存する(ステップ S 8 5)。そして、続いて、元の関数を呼び出す(ステップ S 8 6)。関数からリターンすると、関数定義に構造体の種類指定(structkind_is)の設定が存在するか判別し(ステップ S 8 7)、存在する場合には、設定されている構造体パラメータのデータを種類指定(structkind_is)の設定で定義されているデータ型として解析する(ステップ S 8 8)。そして、リターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内

容をメモリに保存する(ステップ S 8 9)。この処理はユーザから終了の命令があると(ステップ S 9 0)終了される。

【0 0 7 8】

図 3 2 は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 3 0 の定義により取得されたログデータの図である。通常関数定義では、void* となり、ポインタのみしか取得できない構造体のデータが、使用される構造体の種類により、その構造体のデータがログとして取得されている。

【0 0 7 9】

このように本実施形態によれば、通常の方法では取得できないパラメータのログを取得可能になるという効果が得られる。

【0 0 8 0】

【他の実施形態】

なお、本発明は、複数の機器（例えばホストコンピュータ、インタフェイス機器、リーダ、プリンタなど）から構成されるシステムに適用しても、一つの機器からなる装置（例えば、複写機、ファクシミリ装置など）に適用してもよい。

【0 0 8 1】

また、本発明の目的は、前述した実施形態の機能を実現するソフトウェアのプログラムコードを記録した記憶媒体を、システムあるいは装置に供給し、そのシステムあるいは装置のコンピュータ（または CPU や MPU）が記憶媒体に格納されたプログラムコードを読み出し実行することによっても、達成されることは言うまでもない。

【0 0 8 2】

この場合、記憶媒体から読出されたプログラムコード自体が前述した実施形態の機能を実現することになり、そのプログラムコードを記憶した記憶媒体は本発明を構成することになる。

【0 0 8 3】

プログラムコードを供給するための記憶媒体としては、例えば、フロッピー（登録商標）ディスク、ハードディスク、光ディスク、光磁気ディスク、CD-R O

M、CD-R、磁気テープ、不揮発性のメモリカード、ROMなどを用いることができる。

【0084】

また、コンピュータが読出したプログラムコードを実行することにより、前述した実施形態の機能が実現されるだけでなく、そのプログラムコードの指示に基づき、コンピュータ上で稼働しているOS（オペレーティングシステム）などが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

【0085】

さらに、記憶媒体から読出されたプログラムコードが、コンピュータに挿入された機能拡張ボードやコンピュータに接続された機能拡張ユニットに備わるメモリに書込まれた後、そのプログラムコードの指示に基づき、その機能拡張ボードや機能拡張ユニットに備わるCPUなどが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

【0086】

【発明の効果】

以上説明したように本発明によれば、複数のモジュール分けされたソフトウェアの処理ログを容易に取得でき、かつ、ソフトウェアの障害の原因の解析のための工数を削減することが可能となる。

【図面の簡単な説明】

【図1】

本発明の第1の実施形態にかかるログ取得方法を実現するコンピュータ（ソフトウェア評価システム）の構成をあらわす図である。

【図2】

本発明の第1の実施形態にかかるログ取得方法を説明するためにあらわした、関数ロード時の通常のメモリ構成をあらわす図である。

【図3】

本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価シ

システムの I A T P a t c h 使用時のメモリ構成をあらわす図である。

【図 4 A】

本発明の第 1 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの I A T P a t c h を使用時の状態を示す図である。

【図 4 B】

本発明の第 1 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのログ取得処理の流れを示す図である。

【図 5】

本発明の第 1 の実施形態にかかるソフトウェア評価システムの I A T P a t c h を使用時の内部構成をあらわす図である。

【図 6】

本発明の第 1 の実施形態にかかるログ取得方法を説明するためにあらわした、COMサーバのインターフェースのインスタンス作成時の通常のメモリ構成をあらわす図である。

【図 7】

本発明の第 1 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの V T a b l e P a t c h 使用時のメモリ構成をあらわす図である。

【図 8 A】

本発明の第 1 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの V T a b l e P a t c h を使用時の状態を示す図である。

【図 8 B】

本発明の第 1 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのログ取得処理の流れを示す図である。

【図 9】

本発明の第 1 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの、内部構成をあらわす図である。

【図 10】

本発明の第 2 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義の一例を示す図である。

【図 1 1】

本発明の第 2 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いて、ポインタ・パラメータに対してバイナリ取得を指定することで、ポインタ・パラメータのデータの実体をログとして取得する為のIDLによる記述を示す図である。

【図 1 2】

本発明の第 2 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 1 1 の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【図 1 3】

本発明の第 2 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 1 1 の定義により取得されたログデータの図である。

【図 1 4】

本発明の第 3 の本実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いて、コールバック関数等のエクスポートされていない関数をログとして取得するためのIDLによる記述を示す図である。

【図 1 5】

本発明の第 3 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図である。

【図 1 6】

本発明の第 3 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 1 4 の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【図 1 7】

本発明の第 3 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 1 4 の定義を行わない場合に取得されたログデータの図である。

【図 1 8】

本発明の第 3 の実施形態にかかるログ取得方法を実現するソフトウェア評価シ

システムにおいて、図 14 の定義より取得されたログデータの図である。

【図 19】

本発明の第 4 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いて、可変長配列のパラメータをログとして取得するためのIDLによる記述を示す図である。

【図 20】

本発明の第 4 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 19 の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【図 21】

本発明の第 4 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 19 の定義を行わない場合に取得されたログデータ(250)と、図 19 の定義により取得されたログデータ(251)の図である。

【図 22】

本発明の第 5 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、通常関数定義では、パラメータを取得できない関数の例を示す図である。

【図 23】

図 22 に於いて、STRUCTSIZE1, STRUCTSIZE2, STRUCTSIZE3の各構造体がどのようにメモリを使用するかを示す図である。

【図 24】

本発明の第 5 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図 22 の様な関数のパラメータのログを取得するためのIDLによる記述を示す図である。

【図 25】

本発明の第 5 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【図 26】

図 25 で行われる構造体パラメータ解析の詳細をメモリ配置をもとに表した図である。

【図 27】

本発明の第 5 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 24 の定義により取得されたログデータの図である。

【図 28】

本発明の第 6 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、通常の変数定義では、パラメータを取得できない関数の例を示す図である。

【図 29】

図 28 に於いて、STRUCTKIND1, STRUCTKIND2, STRUCTKIND3 の各構造体がどのようなメモリ配置になるかを示した図である。

【図 30】

本発明の第 6 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図 28 の様な関数のパラメータのログを取得するためのIDLによる記述を示す図である。

【図 31】

本発明の第 6 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図 30 の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【図 32】

本発明の第 6 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図 30 の定義により取得されたログデータの図である。

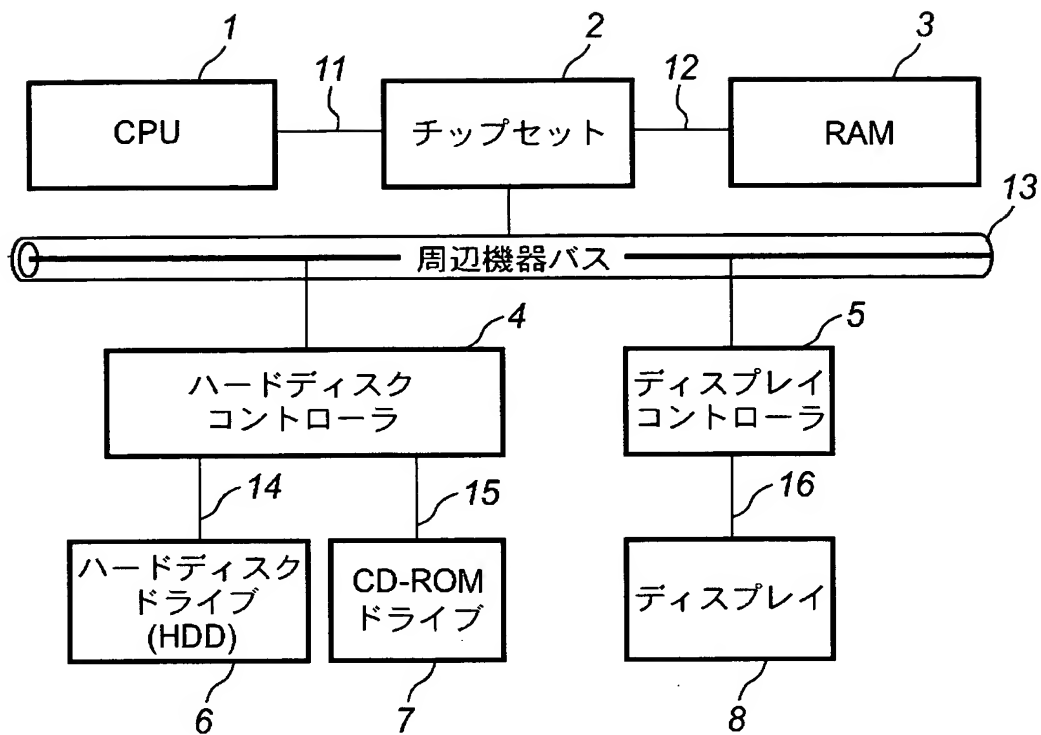
【符号の説明】

- 1：CPU
- 2：チップセット
- 3：RAM
- 4：ハードディスクコントローラ
- 5：ディスプレイコントローラ

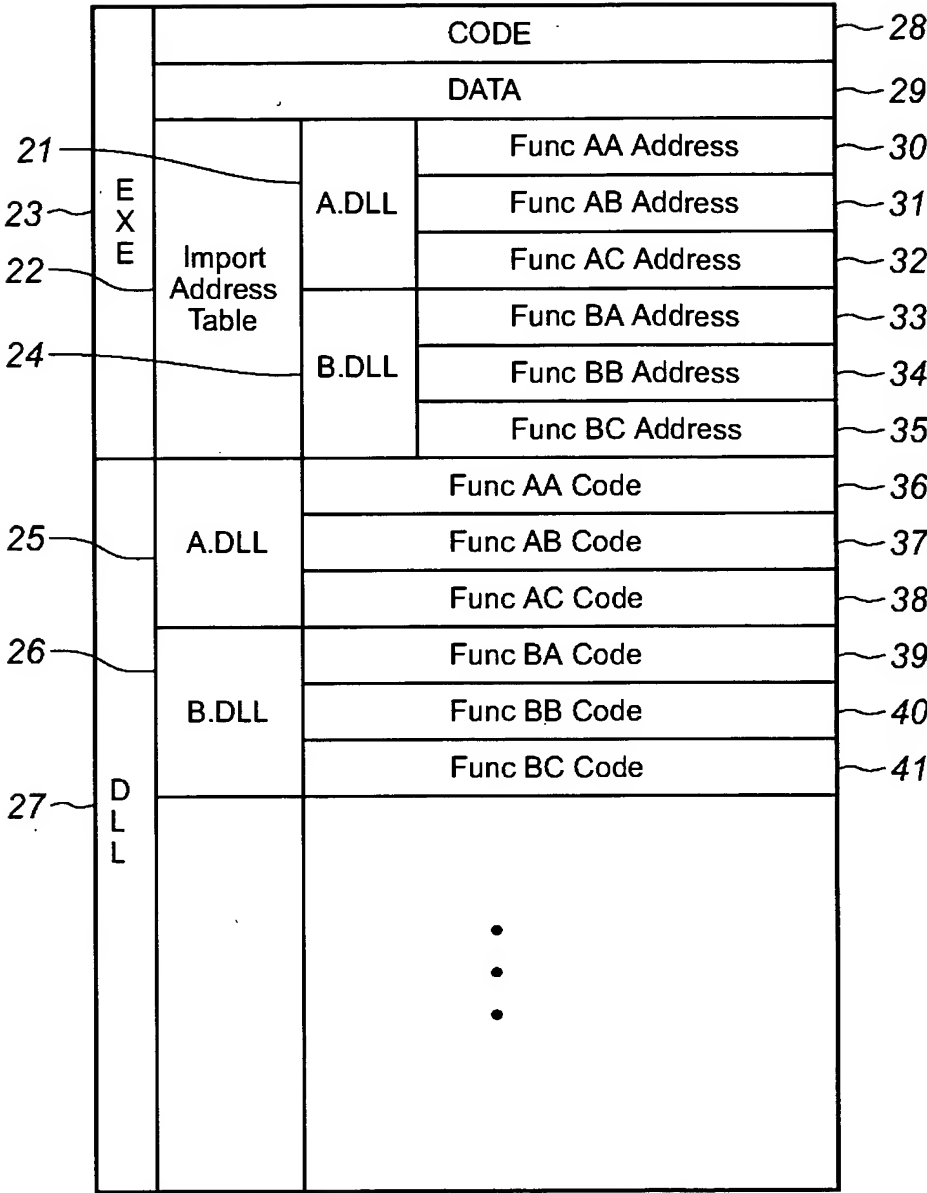
- 6：ハードディスクドライブ
- 7：CD-ROMドライブ
- 8：ディスプレイ
- 11：CPUとチップセットを繋ぐ信号線
- 12：チップセットとRAMを繋ぐ信号線
- 13：チップセットと各種周辺機器とを繋ぐ周辺機器バス
- 14：ハードディスクコントローラとハードディスクドライブを繋ぐ信号線
- 15：ハードディスクコントローラとCD-ROMドライブを繋ぐ信号線
- 16：ディスプレイコントローラとディスプレイを繋ぐ信号線

【書類名】 図面

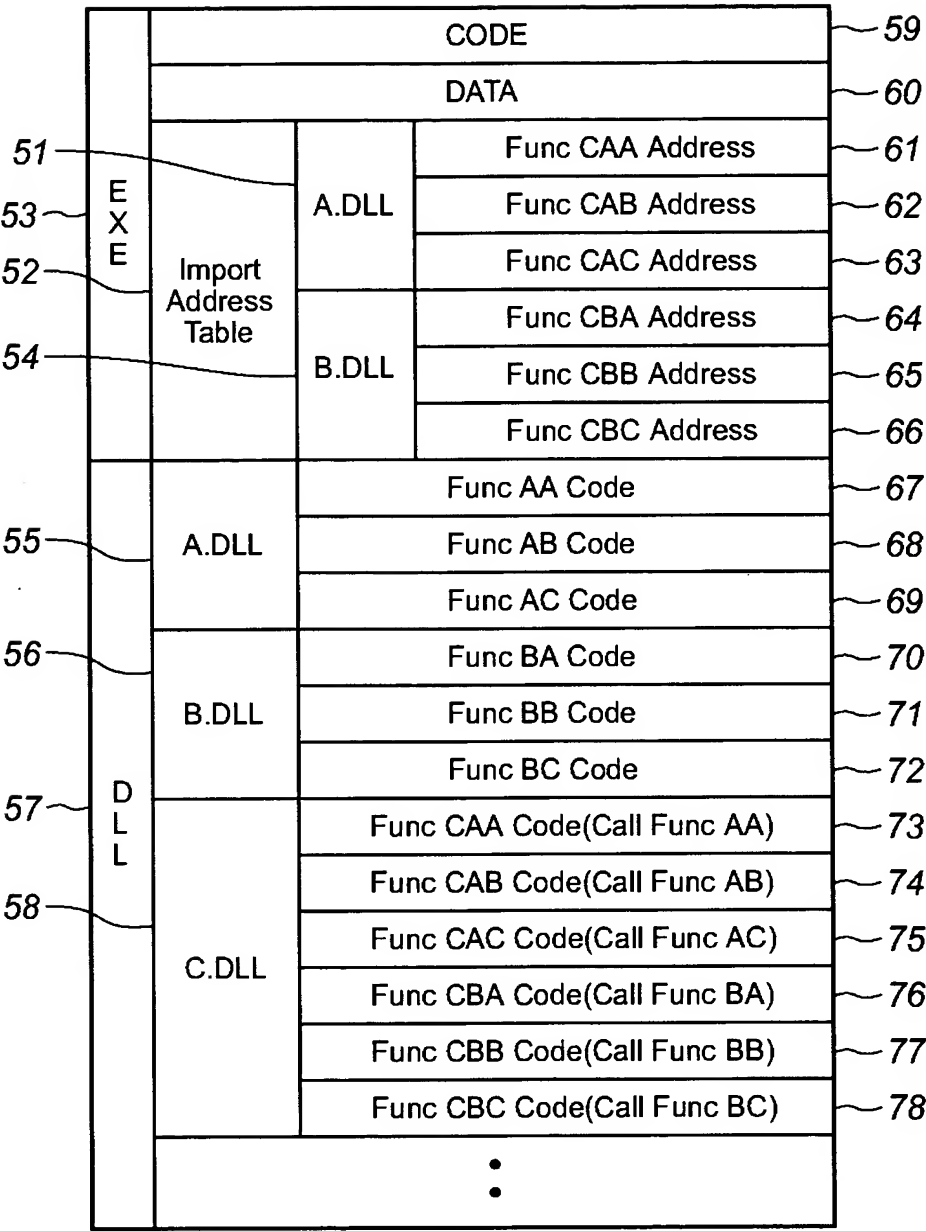
【図 1】



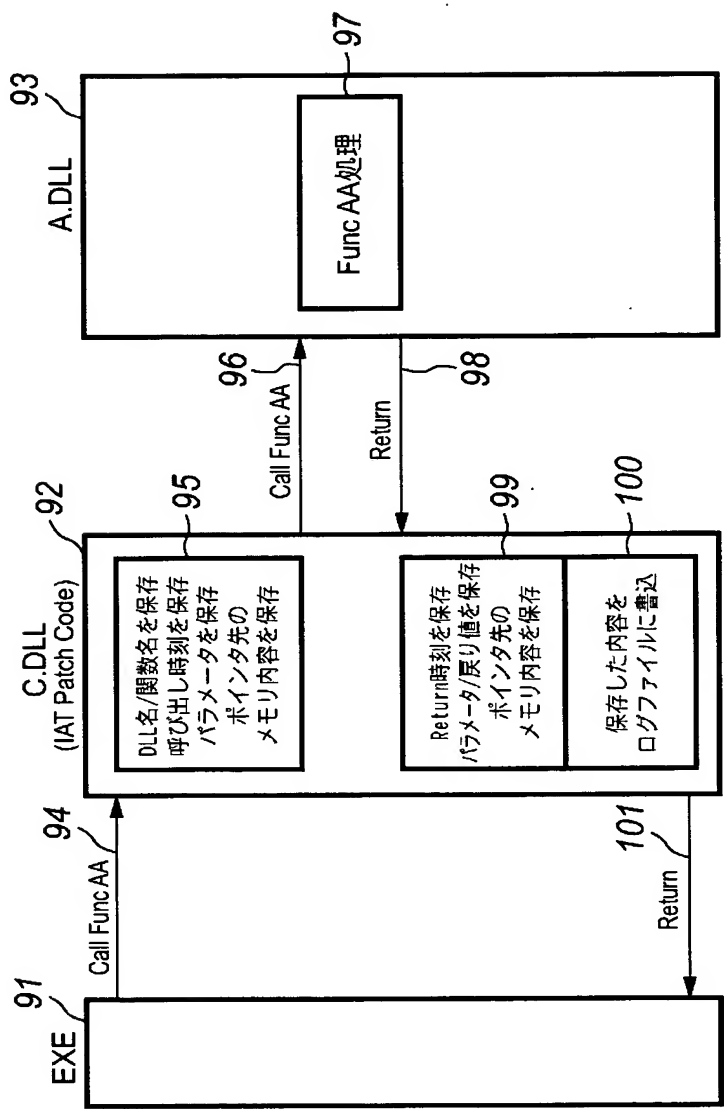
【図 2】



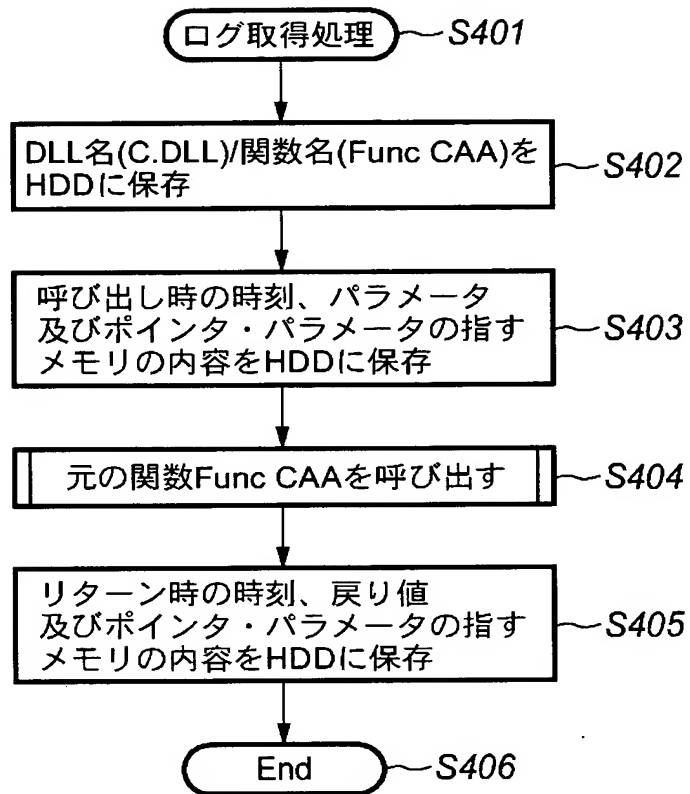
【図 3】



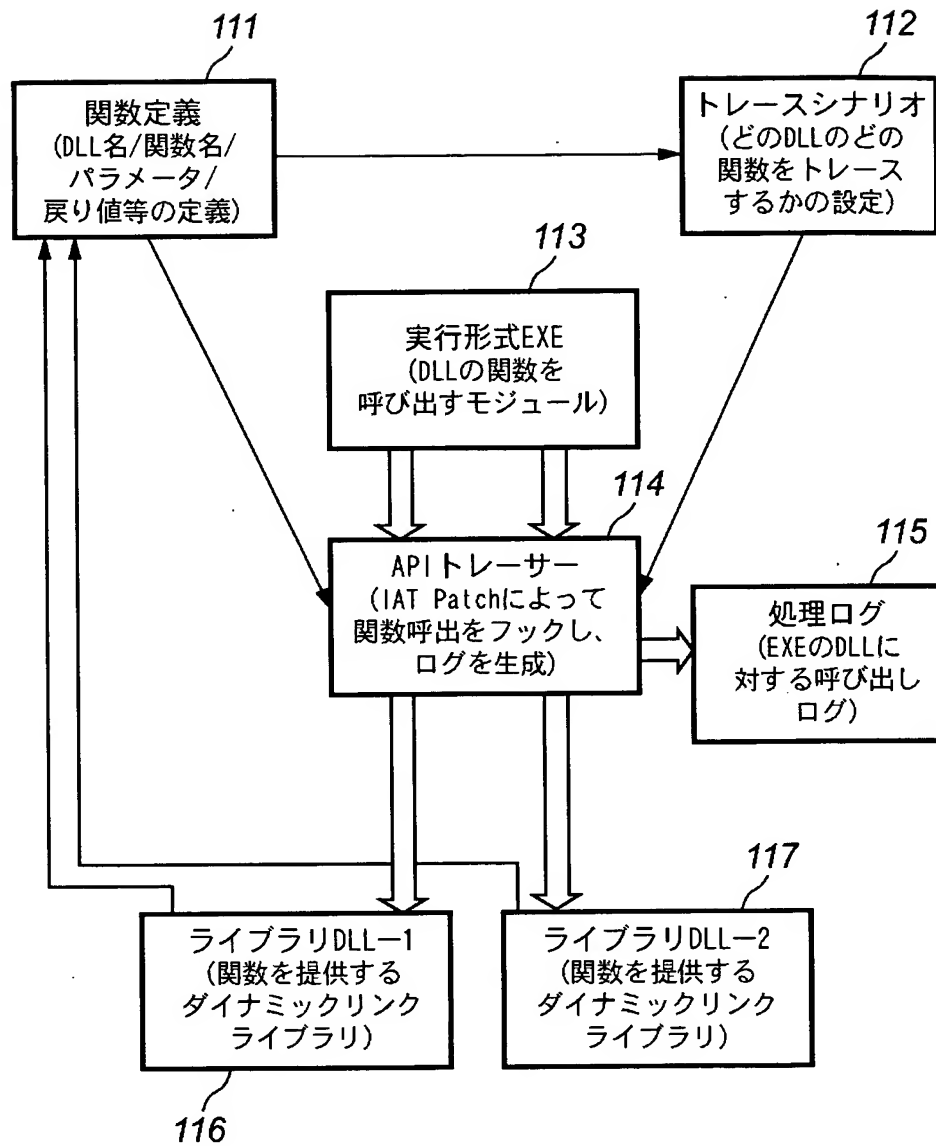
【図 4 A】



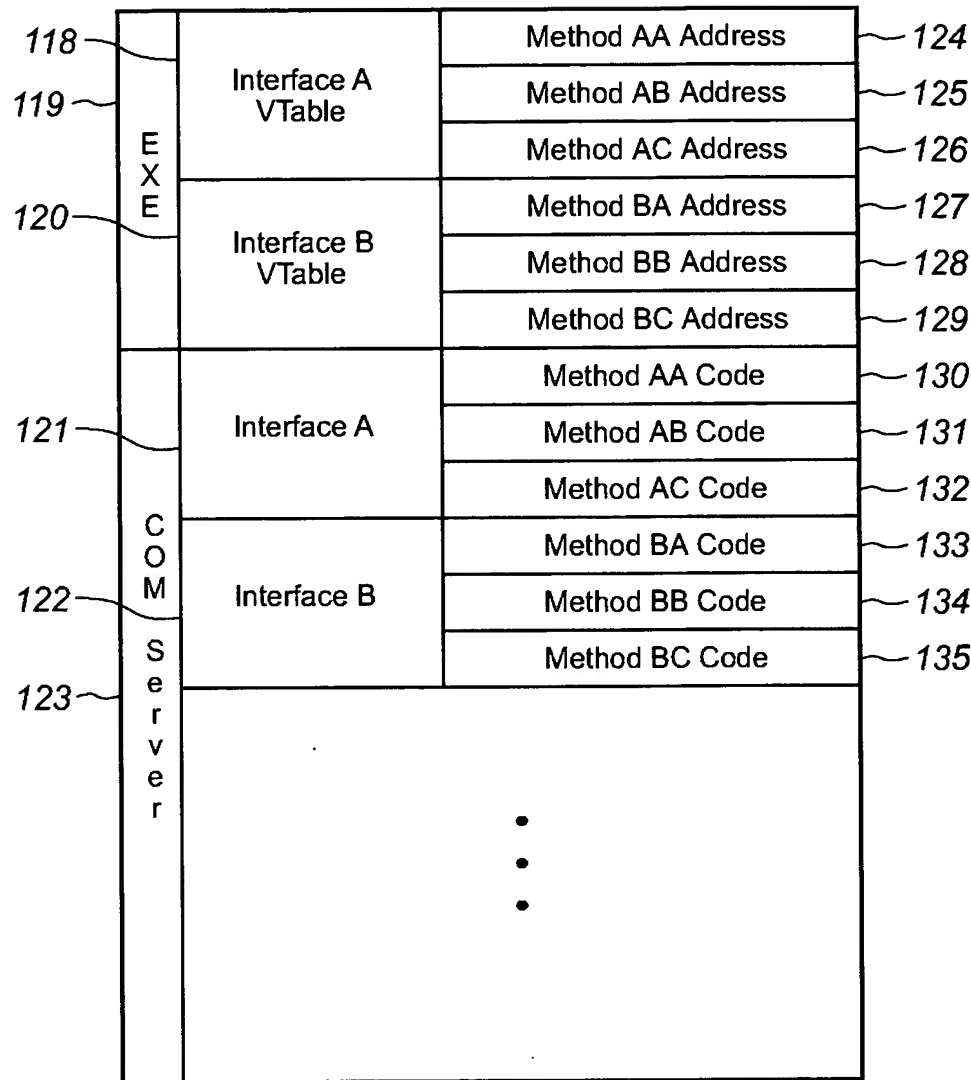
【図 4 B】



【図5】



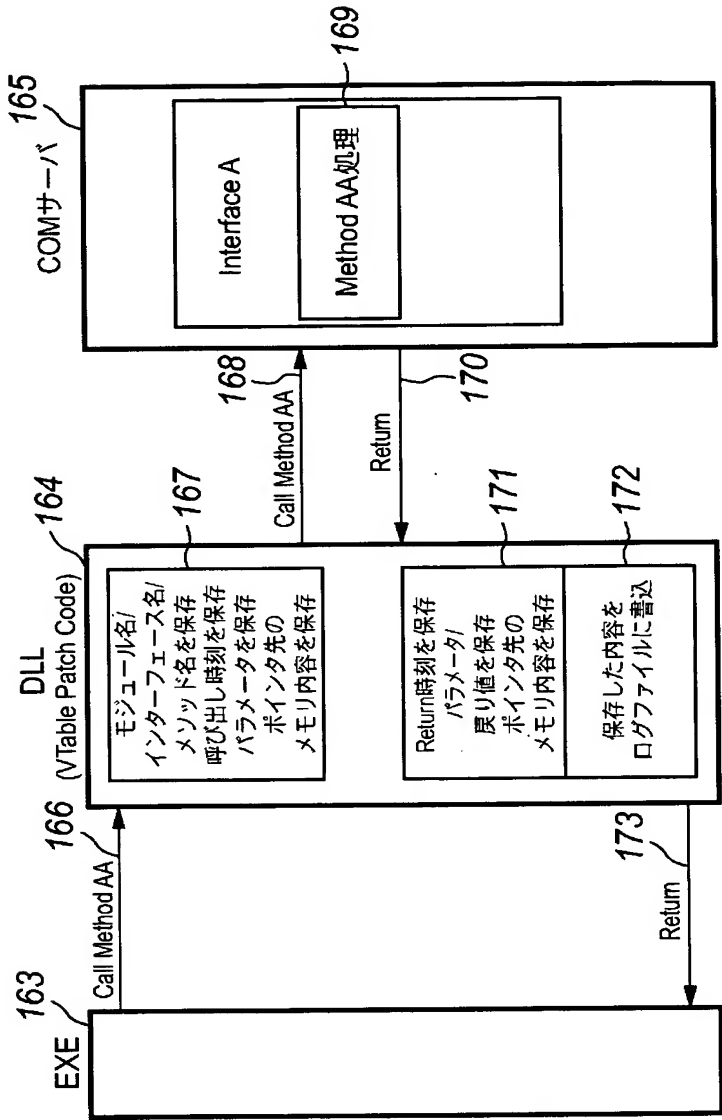
【図 6】



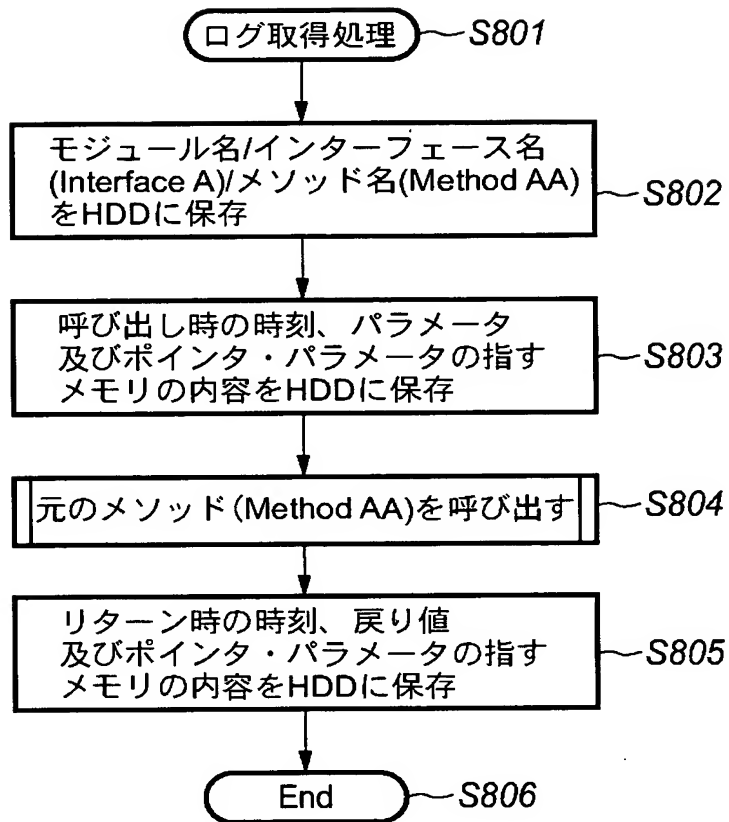
【図 7】

136	E X E	Interface A' VTable	Method A' A Address	145
137			Method A' B Address	146
			Method A' C Address	147
138		Interface B' VTable	Method B' A Address	148
			Method B' B Address	149
			Method B' C Address	150
139	C O M S e r v e r	Interface A	Method AA Code	151
			Method AB Code	152
			Method AC Code	153
141		Interface B	Method BA Code	154
			Method BB Code	155
			Method BC Code	156
140			• •	
142	D L L	Interface A'	Method A' A Code (Call Method AA)	157
			Method A' B Code (Call Method AB)	158
			Method A' C Code (Call Method AC)	159
143		Interface B'	Method B' A Code (Call Method BA)	160
			Method B' B Code (Call Method BB)	161
144			Method B' C Code (Call Method BC)	162

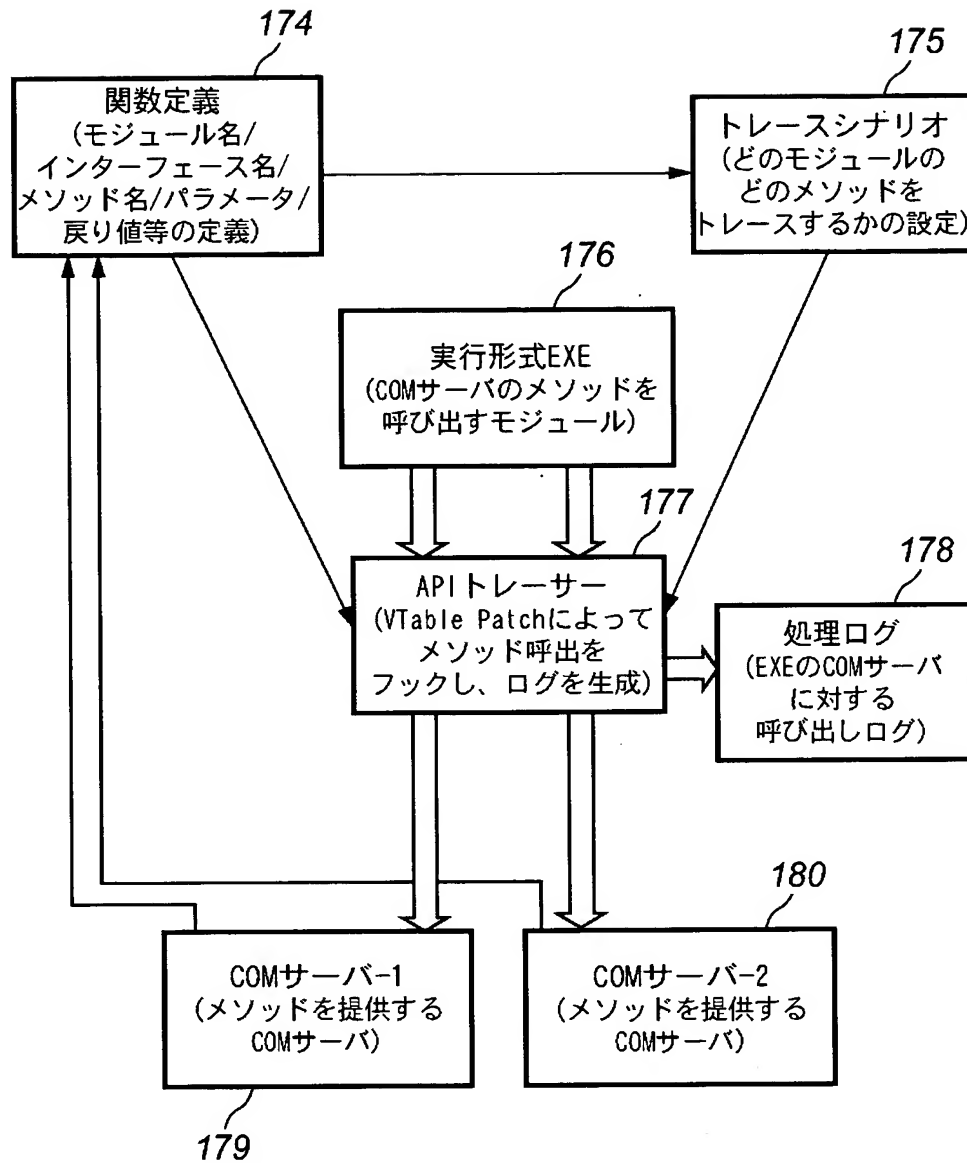
【図 8 A】



【図 8 B】



【図 9】



【図 10】

```
[
    uuid(58DB5633-0694-4340-97CE-4E1AC6BFFBA7),    // TestDllStd
    helpstring("TestDllStd Type Library For PAT"),
    version(1.0)
]

library TestDllStd

    typedef [public] struct
    {
        char chParam;
        unsigned char uchParam;
        short sParam;
        unsigned short usParam;
        int nParam;
        unsigned int unParam;
        long lParam;
        unsigned long ulParam;
        double dbParam;
        float fParam;
    } TESTSTRUCT;
    typedef [public] TESTSTRUCT *LPTESTSTRUCT;
//DEFINE_GUID(GUID_PROGID, 0x8e037d65, 0xefa0, 0x40e7, 0x91, 0x43, 0xef, 0x70, 0x56, 0x94, 0x5b,
0x79);
[
    uuid(8E037D65-EFA0-40e7-9143-EF7056945B79),
] helpstring("TestDllStd.dll for PAT object."),

    interface
    test
    {
        char _stdcall FuncCharStd([in] char chParam);
        char* _stdcall FuncPCharStd([in, out] char* lpchParam);

        TESTSTRUCT _stdcall FuncStructStd([in] TESTSTRUCT TestStruct);
    };
    LPTESTSTRUCT _stdcall FuncPStructStd([in, out] LPTESTSTRUCT lp TestStruct);
}
```

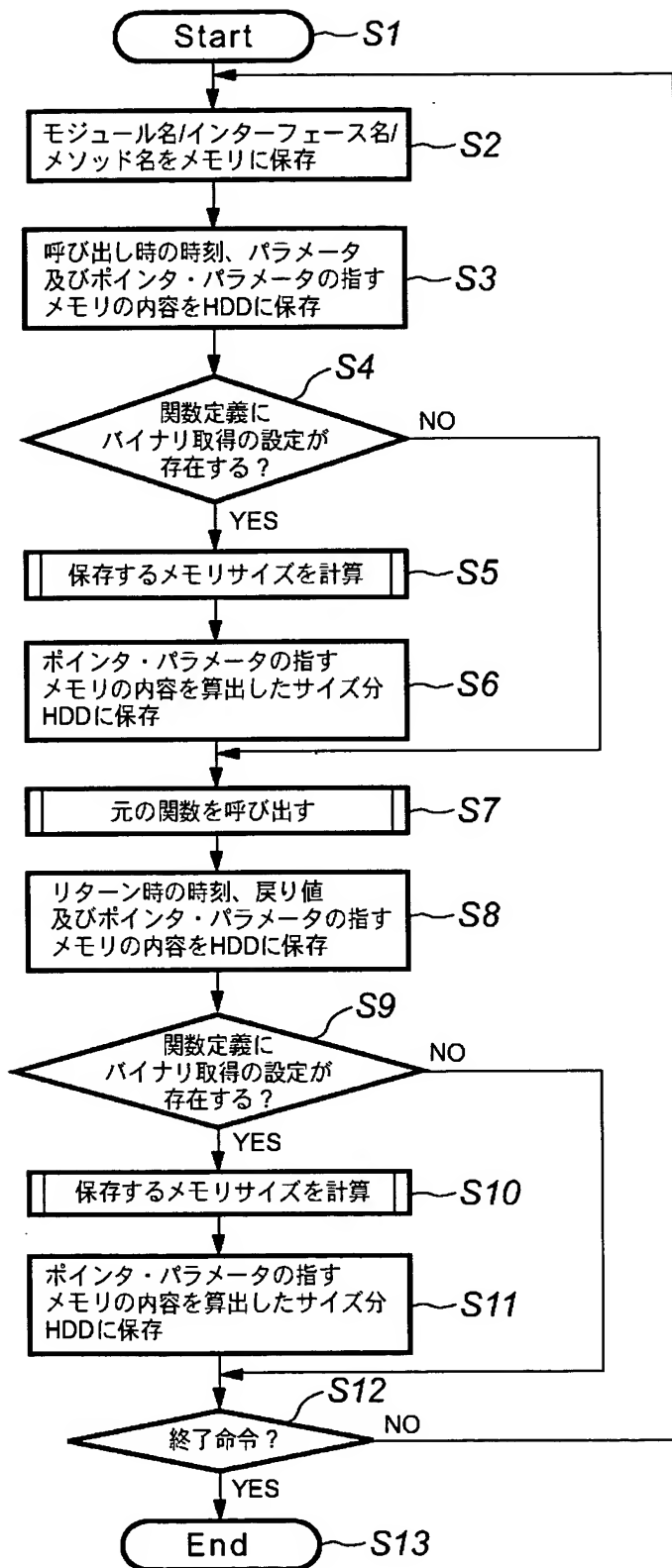
【図 11】

```
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000
interface
test
{
    void_stdcall FuncBinIds
    (
        [out, custom(PAT_PARAM_ATTR_ID, "binid_is({}))"] long* lpParam
    );
    void_stdcall FuncSizes
    (
        [in] DWORD dwCount,
        [out, custom(PAT_PARAM_ATTR_ID, "sizeis_is(dwCount)") int* lpnParam
    );
    void_stdcall FuncLengths
    (
        [in] DWORD dwLength,
        [in, custom(PAT_PARAM_ATTR_ID, "length_is(dwLength)") char* lpszParam
    );
    void_stdcall FuncBytesIs
    (
        [in] DWORD dwSize,
        [in, custom(PAT_PARAM_ATTR_ID, "bytes_is(dwSize)") void* lpParam
    );
    void_stdcall FuncBytesIs2
    (
        [out, custom(PAT_PARAM_ATTR_ID, "bytes_is(12)") void* lpParam
    );
};
```

Diagram illustrating the structure of the `test` interface (labeled 200). The interface defines several methods:

- `FuncBinIds` (labeled 201): A method taking an output parameter of type `long*` (labeled 201).
- `FuncSizes` (labeled 202): A method taking an input parameter of type `DWORD` (`dwCount`) and an output parameter of type `int*` (labeled 202).
- `FuncLengths` (labeled 203): A method taking an input parameter of type `DWORD` (`dwLength`) and an input parameter of type `char*` (labeled 203).
- `FuncBytesIs` (labeled 204): A method taking an input parameter of type `DWORD` (`dwSize`) and an input parameter of type `void*` (labeled 204).
- `FuncBytesIs2` (labeled 205): A method taking an output parameter of type `void*` (labeled 205).

【図 12】



【図 13】

210	モジュール名: TestDllStd. DLL 関数名: FuncBinIdIs 引数 (in): long* lpParam : 0x5034206D/0x10, DataID=0x0001 引数 (out): void: 戻り値: 2002/03/25 22:24:12.025 In時刻: 2002/03/25 22:24:12.035 Out時刻:	211	DataID : 0x0001 Size : 4 00000000 : 10 00 00 00 DataID : 0x0002 Size : 40 00000000 : 05 00 00 00 4A 03 A5 20 00000008 : 06 00 00 00 4B 03 A5 20 00000010 : 07 00 00 00 4C 03 A5 20 00000018 : 08 00 00 00 4D 03 A5 20 00000020 : 09 00 00 00 4E 03 A5 20 DataID : 0x0003 Size : 5 00000000 : 66 4A 70 50 00 DataID : 0x0004 00000000 : 01 5D 66 B2 20 49 20 ...
モジュール名: TestDllStd. DLL 関数名: FuncBinIdIs 引数 (in): long* lpParam : 0x5034206D/0x10, DataID=0x0001 引数 (out): void: 戻り値: 2002/03/25 22:24:12.025 In時刻: 2002/03/25 22:24:12.035 Out時刻:	モジュール名: TestDllStd. DLL 関数名: FuncSizels 引数 (in): DWORD dwCount : 10 引数 (out): int* lpnParam : 0x5034207D/0x5, DataID=0x0002 戻り値: void: In時刻: 2002/03/25 22:24:12.046 Out時刻: 2002/03/25 22:24:12.057	DataID : 0x0001 Size : 4 00000000 : 10 00 00 00 DataID : 0x0002 Size : 40 00000000 : 05 00 00 00 4A 03 A5 20 00000008 : 06 00 00 00 4B 03 A5 20 00000010 : 07 00 00 00 4C 03 A5 20 00000018 : 08 00 00 00 4D 03 A5 20 00000020 : 09 00 00 00 4E 03 A5 20 DataID : 0x0003 Size : 5 00000000 : 66 4A 70 50 00 DataID : 0x0004 00000000 : 01 5D 66 B2 20 49 20
モジュール名: TestDllStd. DLL 関数名: FuncLengthIs 引数 (in): DWORD dwLength : 5 引数 (out): char *lpSzParam : 0x503860C/0x66, DataID=0x0003 戻り値: void: In時刻: 2002/03/25 22:24:12.068 Out時刻: 2002/03/25 22:24:12.079	モジュール名: TestDllStd. DLL 関数名: FuncBytesIs 引数 (in): DWORD dwSize : 7 引数 (out): void* lpParam : 0x503870C/, DataID=0x0004 戻り値: void: In時刻: 2002/03/25 22:24:12.100 Out時刻: 2002/03/25 22:24:12.179 ...	DataID : 0x0003 Size : 5 00000000 : 66 4A 70 50 00 DataID : 0x0004 00000000 : 01 5D 66 B2 20 49 20

【図 14】

```

#define PAT_PARAM_ATTR_ID 00000000-0000-0000-000000000000

typedef [public] struct 220
{
    [in, custom(PAT_PARAM_ATTR_ID, "funcname_is(FuncInternal1)")] DWORD pfnFuncInternal1;
    [in, custom(PAT_PARAM_ATTR_ID, "funcname_is(FuncInternal2)")] DWORD pfnFuncInternal2;
    [in, custom(PAT_PARAM_ATTR_ID, "funcname_is(FuncInternal3)")] DWORD pfnFuncInternal3;
    [in, custom(PAT_PARAM_ATTR_ID, "funcname_is(FuncInternal4)")] DWORD pfnFuncInternal4;
}FUNCPOINTERARRAY;

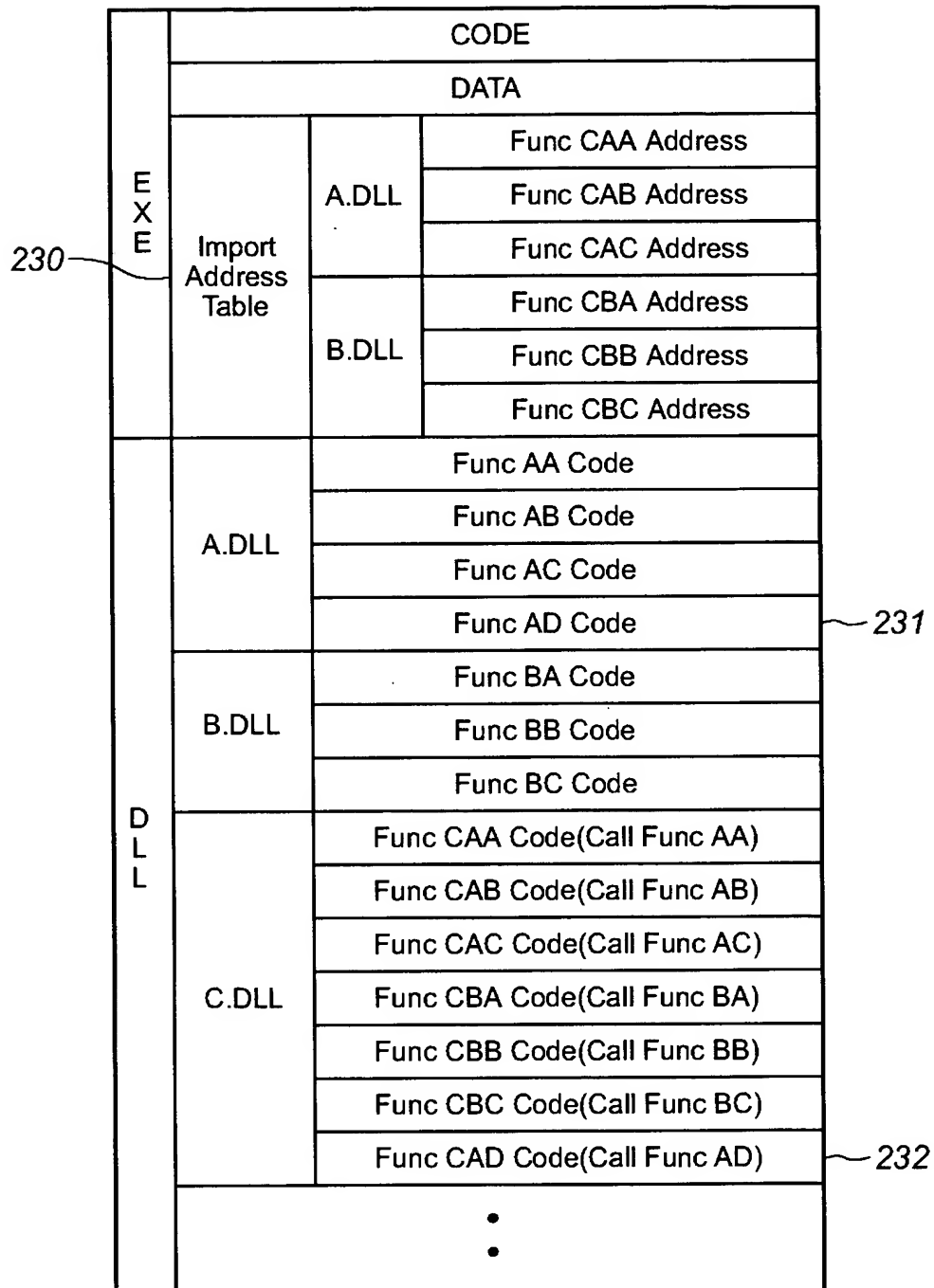
interface
test
{
    void _stdcall SetCallBack
    (
        [in, custom(PAT_PARAM_ATTR_ID, "funcname_is(FuncCallBack)")] DWORD
pfnFuncCallBack
    );
    void FuncCallBack([in] int nParam); 222

    void _stdcall GetFuncPointer
    (
        [out, custom(PAT_PARAM_ATTR_ID, "funcname_is(FuncInternal)")] DWORD
pfnFuncInternal
    );
    void FuncInternal([in, out] char* lpzParam); 224

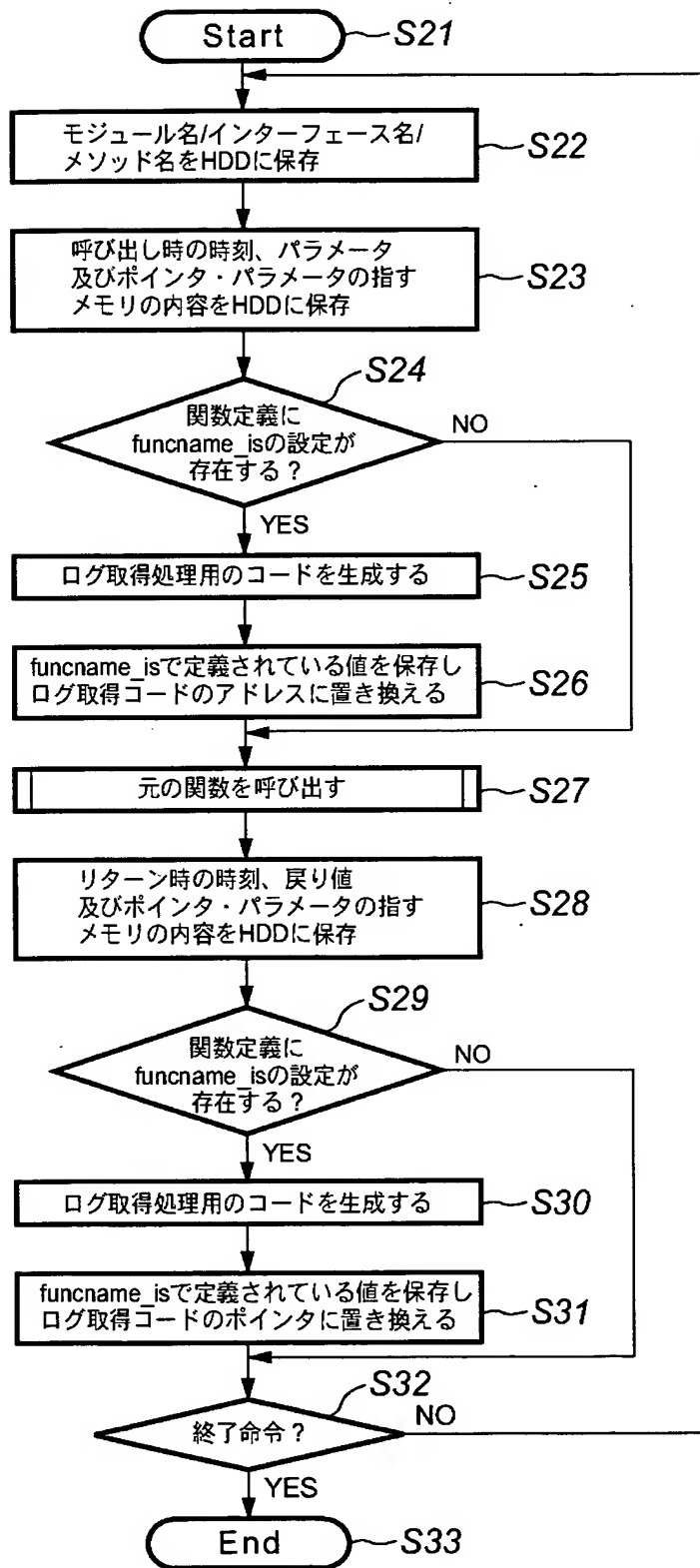
    void _stdcall GetFuncPointerArray
    (
        [out]FUNCPOINTERARRAY* pFuncPointerArray; 225
    );
    void FuncInternal1([in] int nParam);
    void FuncInternal2([in, out] char* lpzaParam); 226
    void FuncInternal3([out] DWORD* dwParam);
    void FuncInternal4();
};

```

【図 1 5】



【図 16】



【図 1 7】

モジュール名：	TestDllStd. DLL
関数名：	FuncSetCallBack
引数 (in)：	DWORD pfnFuncCallBack : 0x0299103F
引数 (out)：	
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 025
Out時刻：	2002/03/25 22 : 24 : 12. 035
モジュール名：	TestDllStd. DLL
関数名：	GetFuncPointer
引数 (in)：	
引数 (out)：	DWORD pfnFuncInternal : 0x029913dF
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 046
Out時刻：	2002/03/25 22 : 24 : 12. 057
モジュール名：	TestDllStd. DLL
関数名：	GetFuncPointerArray
引数 (in)：	
引数 (out)：	FUNCPOINTERARRAY* pFuncPointerArray : 0x503860C DWORD FUNCPOINTERARRAY. pfnFuncInternal1 : 0x02997670 DWORD FUNCPOINTERARRAY. pfnFuncInternal2 : 0x02997708 DWORD FUNCPOINTERARRAY. pfnFuncInternal3 : 0x029977BE DWORD FUNCPOINTERARRAY. pfnFuncInternal4 : 0x0299784F
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 068
Out時刻：	2002/03/25 22 : 24 : 12. 079

【図 1 8】

モジュール名：	TestDllStd. DLL
関数名：	FuncSetCallBack
引数 (in)：	DWORD pfnFuncCallBack : 0x0299103F
引数 (out)：	
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 025
Out時刻：	2002/03/25 22 : 24 : 12. 035
モジュール名：	TestDllStd. DLL
関数名：	FuncCallBack
引数 (in)：	int nParam : 100
引数 (out)：	
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 036
Out時刻：	2002/03/25 22 : 24 : 12. 040
モジュール名：	TestDllStd. DLL
関数名：	GetFuncPointer
引数 (in)：	
引数 (out)：	DWORD pfnFuncInternal : 0x029913dF
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 046
Out時刻：	2002/03/25 22 : 24 : 12. 057
モジュール名：	TestDllStd. DLL
関数名：	FuncInternal
引数 (in)：	char* lpszParam : 0x5038600/0
引数 (out)：	char* lpszParam : 0x5038600/-12
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 060
Out時刻：	2002/03/25 22 : 24 : 12. 065
モジュール名：	TestDllStd. DLL
関数名：	GetFuncPointArray
引数 (in)：	
引数 (out)：	FUNCPOINTERARRAY* pFuncPointerArray : 0x503860C
	DWORD FUNCPOINTERARRAY. pfnFuncInternal1 : 0x02997670
	DWORD FUNCPOINTERARRAY. pfnFuncInternal2 : 0x02997708
	DWORD FUNCPOINTERARRAY. pfnFuncInternal3 : 0x029977BE
	DWORD FUNCPOINTERARRAY. pfnFuncInternal4 : 0x0299784F
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 068
Out時刻：	2002/03/25 22 : 24 : 12. 079
モジュール名：	TestDllStd. DLL
関数名：	FuncInternal4
引数 (in)：	
引数 (out)：	
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 080
Out時刻：	2002/03/25 22 : 24 : 12. 099

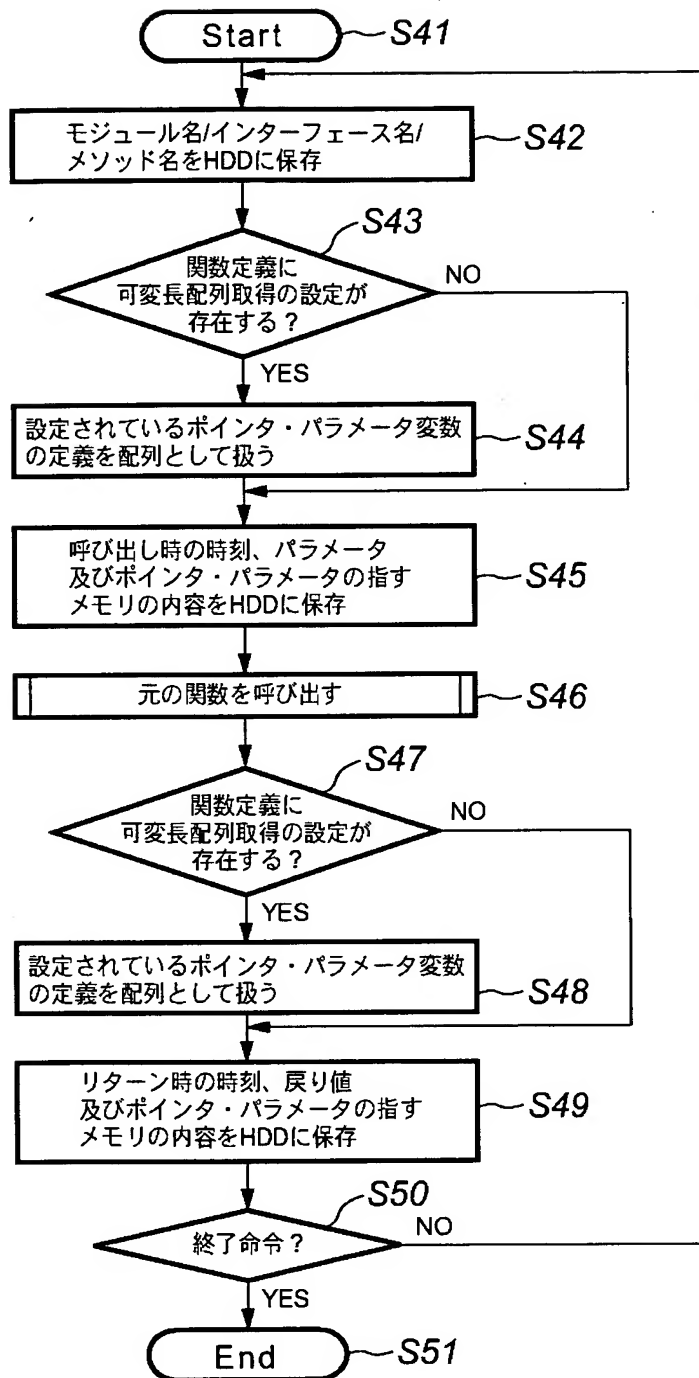
【図 19】

```
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000

interface
test
{
    void_stdcall FuncArrayIs
    (
        [in] DWORD dwCount,
        [in, out, custom(PAT_PARAM_ATTR_ID, "array_is(dwCount)")] int* lpnParam
    );
};
```

240

【図 20】



【図 2 1】

250

モジュール名: TestDllStd. DLL
 関数名: FuncArrayIs
 引数 (in): DWORD dwCount : 4
 int* lpnParam : 0x5034206D/0x00
 引数 (out): int* lpnParam : 0x5034206D/0x01
 戻り値: void :
 In時刻: 2002/03/25 22 : 24 : 12. 025
 Out時刻: 2002/03/25 22 : 24 : 12. 035

...
 モジュール名: TestDllStd. DLL
 関数名: FuncArrayIs
 引数 (in): DWORD dwCount : 3
 int* lpnParam : 0x5034207D/0x00
 引数 (out): int* lpnParam : 0x5034207D/0x05
 戻り値: void :
 In時刻: 2002/03/25 22 : 24 : 12. 046
 Out時刻: 2002/03/25 22 : 24 : 12. 057

251

モジュール名: TestDllStd. DLL
 関数名: FuncArrayIs
 引数 (in): DWORD dwCount : 4
 int* lpnParam : 0x5034206D/Array (int : 0 : 0x00, int : 1 : 0x00, int : 2 : 0x00, int : 3 : 0x00)
 引数 (out): int* lpnParam : 0x5034206D/Array (int : 0 : 0x01, int : 1 : 0x02, int : 2 : 0x03, int : 3 : 0x04)
 戻り値: void :
 In時刻: 2002/03/25 22 : 24 : 12. 025
 Out時刻: 2002/03/25 22 : 24 : 12. 035

...
 モジュール名: TestDllStd. DLL
 関数名: FuncArrayIs
 引数 (in): DWORD dwCount : 3
 int* lpnParam : 0x5034207D/Array (int : 0 : 0x00, int : 1 : 0x00, int : 2 : 0x00)
 引数 (out): int* lpnParam : 0x5034207D/Array (int : 0 : 0x05, int : 1 : 0x10, int : 2 : 0x15)
 戻り値: void :
 In時刻: 2002/03/25 22 : 24 : 12. 046
 Out時刻: 2002/03/25 22 : 24 : 12. 057

【図 2 2】

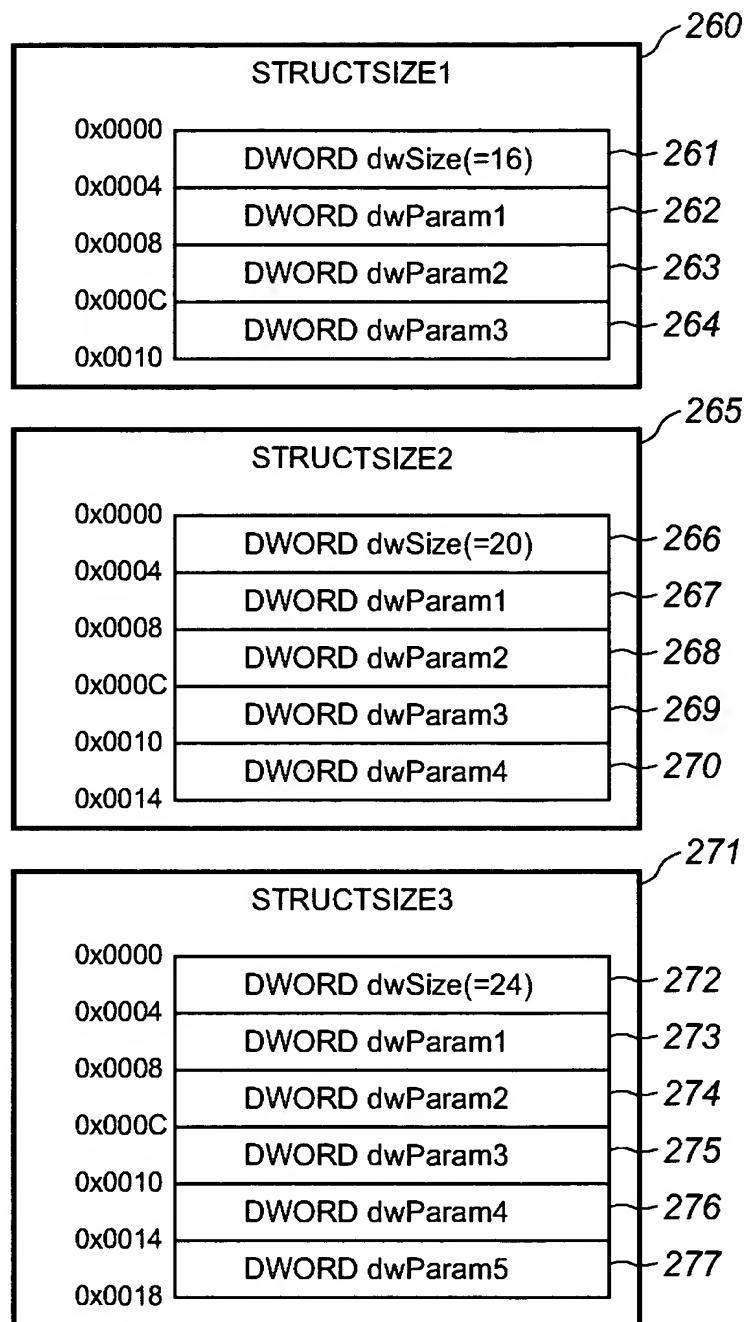
```
typedef struct
{
    DWORD dwSize;
    DWORD dwParam1;
    DWORD dwParam2;
    DWORD dwParam3;
}STRUCTSIZE1;

typedef struct
{
    DWORD dwSize;
    DWORD dwParam1;
    DWORD dwParam2;
    DWORD dwParam3;
    DWORD dwParam4;
}STRUCTSIZE2;

typedef struct
{
    DWORD dwSize;
    DWORD dwParam1;
    DWORD dwParam2;
    DWORD dwParam3;
    DWORD dwParam4;
    DWORD dwParam5;
}STRUCTSIZE3;

void FuncGetData (DWORD dwKind, void* lpBuf)
{
    switch(dwKind)
    {
    case 1:
        //lpBufをSTRUCTSIZE1のポインタとして処理
        break;
    case 2:
        //lpBufをSTRUCTSIZE2のポインタとして処理
        break;
    case 3:
        //lpBufをSTRUCTSIZE3のポインタとして処理
        break;
    }
}
```

【図 23】



【図 24】

```
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000

typedef [public] struct
{
    [custom (PAT_PARAM_ATTR_ID,"structsize_is()")]DWORD dwSize;
    DWORD dwParam1;
    DWORD dwParam2;
    DWORD dwParam3;
    DWORD dwParam4;
    DWORD dwParam5;
}STRUCTSIZE;

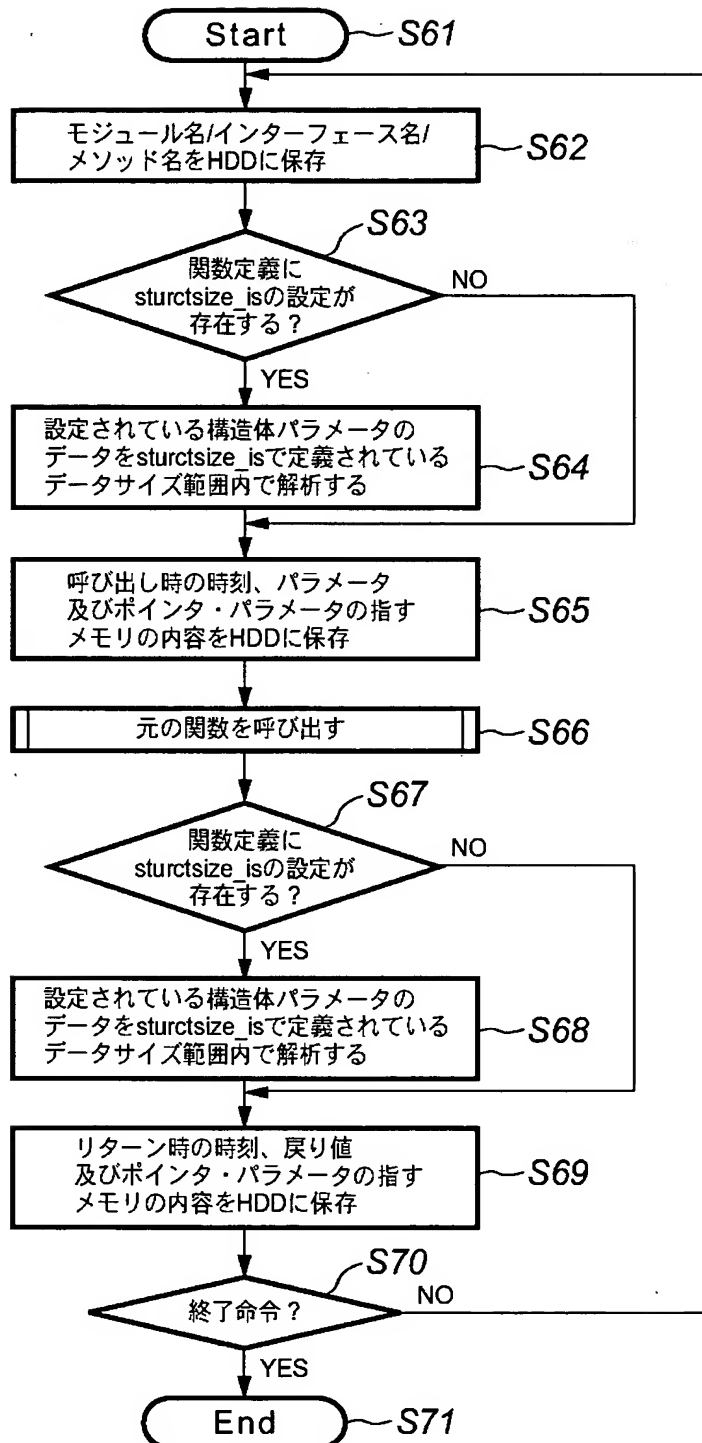
interface
test
{
    void FuncGetData
    (
        [in] DWORD dwKind,
        [in, out] STRUCTSIZE* lpBuf
    );
};
```

290

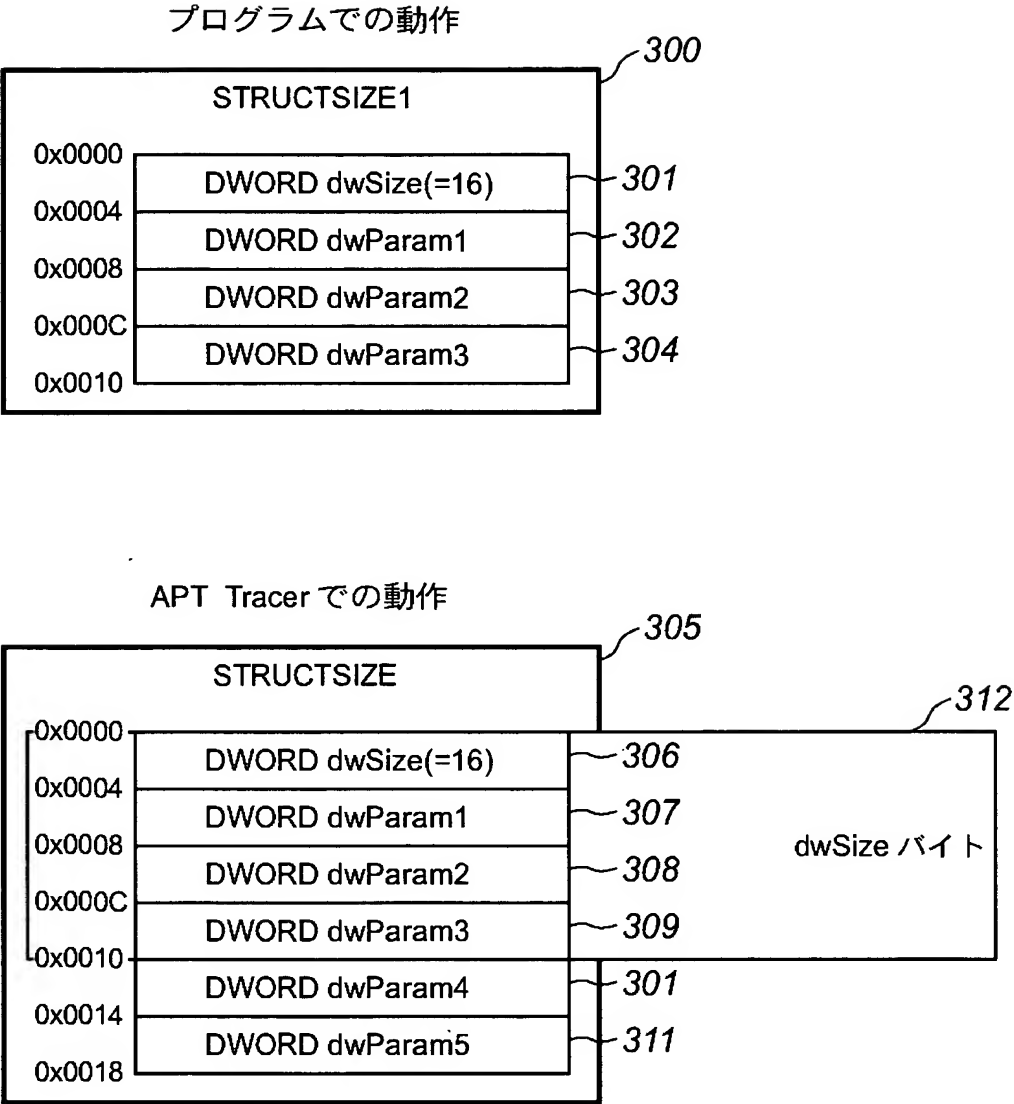
291

292

【図 25】



【図 2 6】



【図 2 7】

モジュール名 :	TestDllStd. DLL
関数名 :	FuncGetData
引数 (in) :	DWORD dwKind : 1 STRUCTSIZE* pBuf : 0x503860C DWORD STRUCTSIZE. dwSize : 16 DWORD STRUCTSIZE. dwParam1 : 0 DWORD STRUCTSIZE. dwParam2 : 0 DWORD STRUCTSIZE. dwParam3 : 0
引数 (out) :	STRUCTSIZE* pBuf : 0x503860C DWORD STRUCTSIZE. dwSize : 16 DWORD STRUCTSIZE. dwParam1 : 1 DWORD STRUCTSIZE. dwParam2 : 2 DWORD STRUCTSIZE. dwParam3 : 3
戻り値 :	void :
In時刻 :	2002/03/25 22 : 24 : 12. 025
Out時刻 :	2002/03/25 22 : 24 : 12. 035
モジュール名 :	TestDllStd. DLL
関数名 :	FuncGetData
引数 (in) :	DWORD dwKind : 3 STRUCTSIZE* pBuf : 0x503990C DWORD STRUCTSIZE. dwSize : 24 DWORD STRUCTSIZE. dwParam1 : 0 DWORD STRUCTSIZE. dwParam2 : 0 DWORD STRUCTSIZE. dwParam3 : 0 DWORD STRUCTSIZE. dwParam4 : 0 DWORD STRUCTSIZE. dwParam5 : 0
引数 (out) :	STRUCTSIZE* pBuf : 0x503990C DWORD STRUCTSIZE. dwSize : 24 DWORD STRUCTSIZE. dwParam1 : 10 DWORD STRUCTSIZE. dwParam2 : 20 DWORD STRUCTSIZE. dwParam3 : 30 DWORD STRUCTSIZE. dwParam4 : 40 DWORD STRUCTSIZE. dwParam5 : 50
戻り値 :	void :
In時刻 :	2002/03/25 22 : 24 : 12. 046
Out時刻 :	2002/03/25 22 : 24 : 12. 057
...	

【図 2 8】

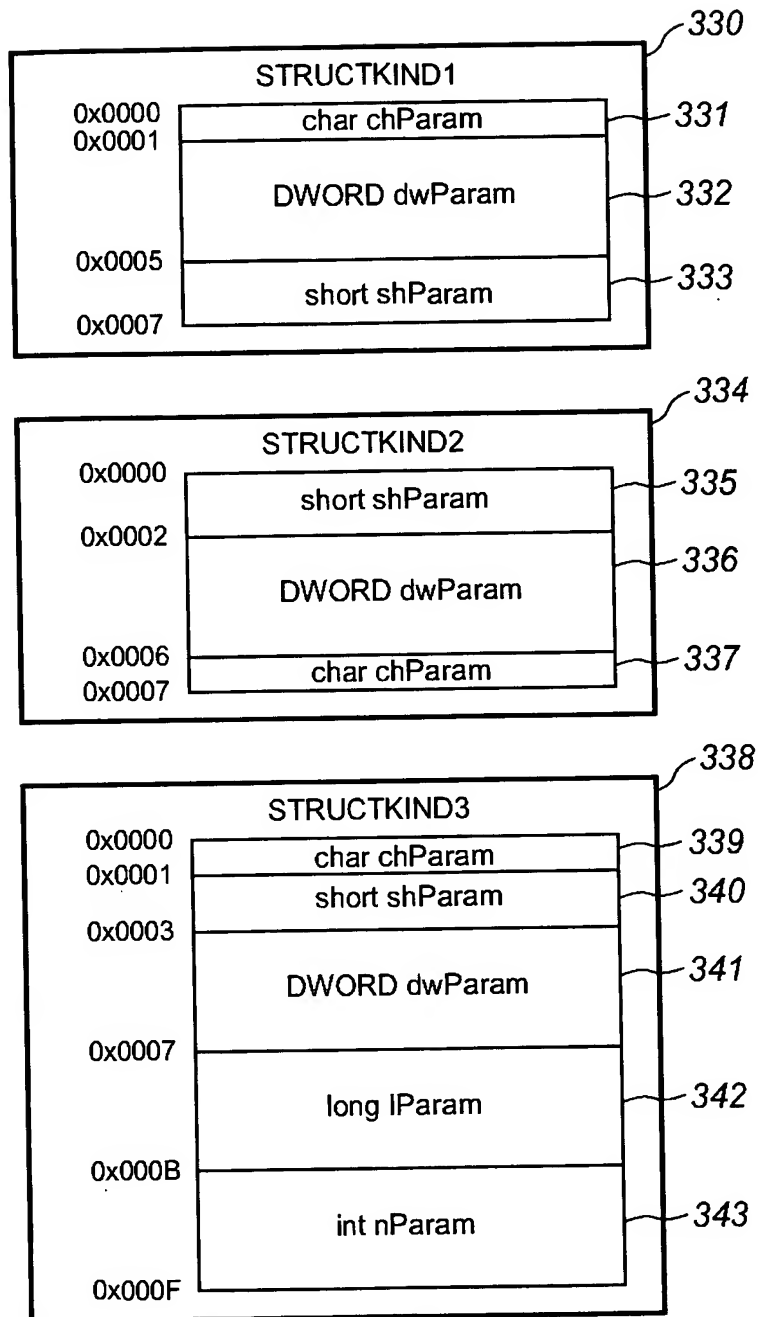
```
typedef struct
{
    char chParam;
    DWORD dwParam;
    short shParam;
}STRUCTKIND1;

typedef struct
{
    short shParam;
    DWORD dwParam;
    char chParam;
}STRUCTKIND2;

typedef struct
{
    char chParam;
    DWORD dwParam;
    short shParam;
    long lParam;
    int nParam;
}STRUCTKIND3;

void FuncGetData (DWORD dwKind, void* lpBuf)
{
    switch(dwKind)
    {
    case 1:
        //lpBufをSTRUCTKIND1のポインタとして処理
        break;
    case 2:
        //lpBufをSTRUCTKIND2のポインタとして処理
        break;
    case 3:
        //lpBufをSTRUCTKIND3のポインタとして処理
        break;
    }
}
```

【図 29】



【図 30】

```
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-000000000000

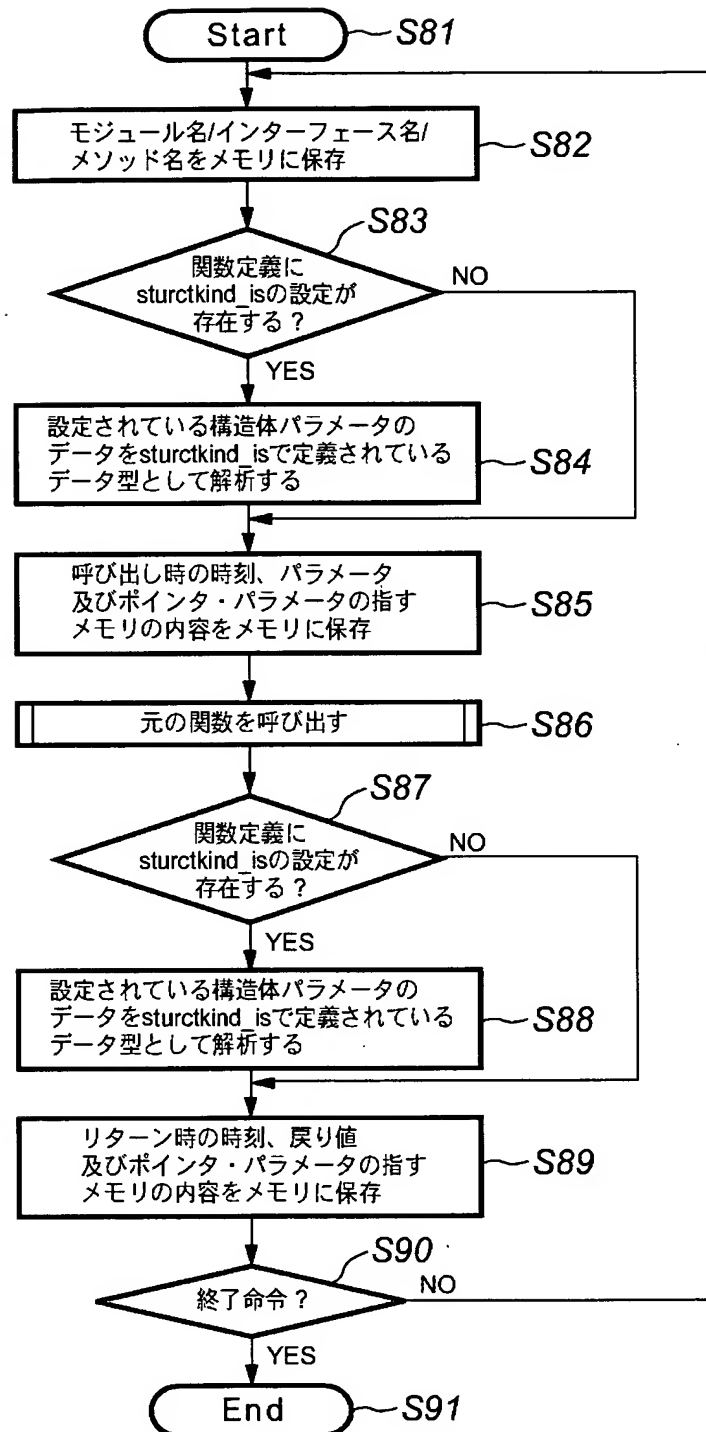
typedef [public] struct
{
    char chParam;
    DWORD dwParam;
    short shParam;
}STRUCTKIND1;

typedef [public] struct
{
    short shParam;
    DWORD dwParam;
    char chParam;
}STRUCTKIND2;

typedef [public] struct
{
    char chParam;
    short shParam;
    DWORD dwParam;
    long lParam;
    int nParam;
}STRUCTKIND3;

interface
test
{
    void FuncGetData
    (
        [in]DWORD dwKind,
        [in, out, custom(PAT_PARAM_ATTR_ID,
        "structKind_is (dwKind : 1: STRUCTKIND1*, 2: STRUCTKIND2*, 3: STRUCTKIND3* )")]
        void* lpBuf
    );
};
```

【図 31】



【図 3 2】

モジュール名 :	TestDllStd. DLL
関数名 :	FuncGetData
引数 (in) :	DWORD dwKind : 1 STRUCTKIND1* pBuf : 0x503860C char STRUCTKIND1. chParam : 0 DWORD STRUCTKIND1. dwParam : 0 short STRUCTKIND1. shParam : 0
引数 (out) :	STRUCTKIND1* pBuf : 0x503860C char STRUCTKIND1. chParam : 1 DWORD STRUCTKIND1. dwParam : 2 short STRUCTKIND1. shParam : 3
戻り値 :	void :
In時刻 :	2002/03/25 22 : 24 : 12. 025
Out時刻 :	2002/03/25 22 : 24 : 12. 035
モジュール名 :	TestDllStd. DLL
関数名 :	FuncGetData
引数 (in) :	DWORD dwKind : 3 STRUCTKIND3* pBuf : 0x503990C char STRUCTKIND3. chParam : 0 short STRUCTKIND3. shParam : 0 DWORD STRUCTKIND3. dwParam : 0 long STRUCTKIND3. lParam : 0 int STRUCTKIND3. nParam : 0
引数 (out) :	STRUCTKIND3* pBuf : 0x503990C char STRUCTKIND3. chParam : 10 short STRUCTKIND3. shParam : 20 DWORD STRUCTKIND3. dwParam : 30 long STRUCTKIND3. lParam : 40 int STRUCTKIND3. nParam : 50
戻り値 :	void :
In時刻 :	2002/03/25 22 : 24 : 12. 046
Out時刻 :	2002/03/25 22 : 24 : 12. 057

...

【書類名】 要約書

【要約】

【課題】 ソフトウェアの処理ログを容易に取得でき、かつ、バグの解析のための工数を削減することが可能なログ取得方法を提供する。

【解決手段】 関数 (FuncAA) を備えるプログラム 9 1 において、実行中のログを取得するログ取得方法であって、ロードされた関数 (FuncAA) のアドレスを、ログ取得のための関数 (9 2) のアドレスに書き換える工程を備え、ログ取得のための関数 (9 2) は、関数 (FuncAA) のアドレスを呼び出し (9 6)、該所定の処理を実行させ (9 7)、受け取った実行結果 (9 8) をプログラム 9 1 に渡す工程 (1 0 1) と、プログラム 9 1 内の関数定義に、ポインタ・パラメータが所定の定義方法で定義されているか否かを判断する工程と、定義されていた場合、該定義方法に基づいて該ポインタ・パラメータの指すメモリの内容をログとして記録する工程とを備える。

【選択図】 図 4 A

特願 2 0 0 2 - 1 9 1 1 2 8

出 願 人 履 歴 情 報

識別番号

[0 0 0 0 0 1 0 0 7]

1. 変更年月日

1 9 9 0 年 8 月 3 0 日

[変更理由]

新規登録

住 所

東京都大田区下丸子 3 丁目 3 0 番 2 号

氏 名

キャノン株式会社